Programmieren

in Maschinensprache mit dem

Commodore-64

Eine Einführung mit vielen Beispielen



C. Lorenz

ISBN 3–921682–70–3

2. völlig neu überarbeitete Auflage 1984

Gedruckt in der Bundesrepublik Deutschland – Printed in West-Germany – Imprime'en RFA.

C. Lorenz

Programmieren
in Maschinensprache mit dem
Commodore-64

C. Lorenz

Programmieren
in Maschinensprache mit dem Commodore-64

Eine Einführung mit
vielen Beispielen

Vorwort

Programmierung in Assemblersprache mit dem C-64.

Lernen durch Anwendung.

Viele der Anfänger unter Ihnen haben die Handbücher Ihres C-64 schon gründlich studiert und sicher viele Artikel oder sogar Bücher über Klein-Computer gelesen. Viele von Ihnen haben Ihren C-64 schon in BASIC,PASCAL oder FORTH programmiert. Nach einer gewissen Zeit haben Sie jedoch sicher festgestellt,dass diese Sprachen fuer die eine oder andere Anwendung viel zu langsam sind. Man denke nur an die Programmierung von Bewegungsabläufen, Grafik und Sound. Weiterhin wollen Sie jetzt sicher auch endlich mehr darueber wissen, was eigentlich im Inneren des Computers vorgeht. Sie kennen bereits die Grundlagen und die Grundzüge der binären Rechentechnik. Wie hängen diese Zahlen mit den Informationen zusammen, die auf dem Bildschirm dargestellt werden? Wie kann ich hier selbst eingreifen und in Maschinensprache programmieren? Auf all diese Fragen wollen wir in diesem Buch eine Antwort geben. Der Zweck dieses Buches ist, Ihnen zu zeigen, wie der Commodore 64 in 6510/6502 Maschinensprache programmiert werden kann. Sie können einen Maschinensprachenmonitor wie z.B. den Supermon 64, den in diesem Buche enthaltenen einfachen Monitor oder auch den 64Mon von Commodore verwenden. Die Programme können auch mit Hilfe des im MACROFIRE eingebauten Monitors eingegeben werden. Wer den Assembler Quelltext eingeben will, braucht einen symbolischen Assembler wie z.B. MACROFIRE oder ein ähnliches Produkt.

Inhaltsverzeichnis

ZU DIESEM BUCH

Dieses Buch soll Ihnen den Einstieg in die Maschinensprachenprogrammierung so einfach wie moeglich machen. Wir haben dabei grossen Wert darauf gelegt, Ihnen ein komplettes Paket anzubieten. Aus diesem Grunde enthaelt dieses Buch nicht nur eine sehr einfache, didaktisch geschickt gemachte Einfuehrung, sondern darueber hinaus auch das notwendige Monitorprogramm zum Eingeben und Starten der in diesem Buche enthaltenen Programme.
Wer noch einen Schritt weiter gehen will, findet sogar einen kleinen Assembler in diesem Buch.
Dieser kleine Assembler dient in erster Linie zum Erlernen der Assemblerprogrammierung. Er ist sehr praktisch, kann jedoch nicht an die Adresse $C000 assemblieren. (nicht ueber 32000 dez)
Die in diesem Buche enthaltenen Listing wurden jedoch mit dem Editor/Assembler "MACROFIRE" eingegeben und assembliert. Wer also etwas professioneller arbeiten moechte, dem sei ein symbolischer Assembler empfohlen.
Am Schluss des Buches haben wir die Grundlagen fuer den Maschinen Sprachen Programmierer, wie Zahlensysteme, binaere Rechentechnik usw. kurz abgehandelt. Wer also hier noch nicht ganz sattelfest ist,beginnt zunaechst mit dem Anhang A.Alle anderen Leser koennen gleich mit Beispielen in die wunderbare Welt der Maschinensprache des 6510/6502 einsteigen.
Wie immer hat sich der Autor sehr viel Muehe gegeben und alle Programme getestet. Sollten sich trotzdem Fehler eingeschlichen haben, so waeren wir Ihnen fuer jeden Vorschlag dankbar. Alle Programme aus diesem Buche haben wir auch fuer Sie auf Diskette bereitgestellt. Sie sind als Quelltext fuer MACROFIRE abgelegt.Die Diskette ohne MACROFIRE kostet DM 99.- Sie sparen sich damit das laestige eintippen. Uebrigens, der Monitor und der kleine Assembler sind auch mit auf der Diskette enthalten.
Wir wuenschen Ihnen bei Ihren Ausfluegen ins Reich der Maschinensprache viel Erfolg und hoffen, dass Sie neben viel Freude auch einen Nutzen fuer Ihre berufliche Weiterbildung daraus ziehen koennen.

Holzkirchern/Obb. Winfried Hofacker

1

Teil 1

Das Vorhandensein der Programmiersprache BASIC läßt die meisten Programmierer vergessen, daß in dem Rechner Ausdrücke wie IF THEN usw. eine Folge von Bitmustern sind, welche die CPU als ein ausführbares Maschinenprogramm liest. Sobald man aber die Sprache BASIC verläßt, muß man selbst auf diese Ebene des Programmierens in Maschinencode heruntersteigen. Dies ist vor allem dann notwendig, wenn der Rechner u. a. zu Steuerungsaufgaben in Verbindung mit der Außenwelt eingesetzt wird oder Systemprogramme entwickelt werden sollen. Diese Einführung ist also für alle diejenigen gedacht, die sich bisher mit BASIC befasst haben, und die nun etwas tiefer in die Programmierung einsteigen wollen. Daß hierbei wiederum der Prozessor 6510/6502 mit seinem Befehlsvorrat im Vordergrund steht, liegt einfach daran, daß die meisten, auch die neuesten, Home-Computer diese CPU verwenden. Es ist aber gleichgültig, mit welchem Prozessor man das Programmieren in Maschinensprache lernt. Das Umsteigen auf einen anderen Prozessor bedeutet nur die Verwendung eines anderen Befehlsvorrates, das "Denken in Maschinensprache" bleibt das Gleiche. In diesem ersten Teil sollen nun zuerst einige Grundbegriffe erläutert werden.

Der Einstieg in die Maschinensprache erfolgt über den MONITOR. Dies ist das Betriebssystem, welches nach dem Einschalten des Rechners aktiv wird und von sich aus das weitere Einlesen und Ausführen von Programmen übernimmt. Dieser Monitor ist für das Programmieren in Maschinencode sehr wichtig. Einmal erkennt er Monitorbefehle wie z. B. Ausgeben des Speicherinhaltes auf den Bildschirm oder Starten des Programms.

Andererseits enthält er aber Unterprogramme, die in den eigenen Programmen verwendet werden können. Die am häufigsten gebrauchten Unterprogramme sind die Ausgabe eines Zeichens auf ein Ausgabemedium und die Eingabe eines Zeiches in den Rechner.

Der Einstieg in den Monitor erfolgt zum Beispiel beim APPLE II mit Call-151 aus BASIC, beim OHIO C1P durch Eingabe von M nach dem Einschalten und der AIM 65 ist nach dem Einschalten automatisch im Monitor. Der ATARI 400/800 befindet sich im EDIT-Mode, wenn der Assembler/Editor verwendet wird. Die Beispiele in diesem Buch wurden mit einem symbolischen Editor/Assembler geschrieben.Der Hexcode kann auch mit dem in diesem Buch enthaltenen Monitor eingegeben werden.

Ein Maschinenprogramm ist eng mit dem Speicher des Computers verknüpft. Jeder Befehl ist an einer festen Adresse gespeichert. Die Adresse ist die Hausnummer jedes Speicherplatzes im Rechner. Wichtig für die Programmausführung ist die Startadresse des Programms. Dies ist die Adresse des Speicherplatzes, in welcher der zuerst auszuführende Befehl gespeichert ist. Diese Adresse wird durch einen Monitorstart-Befehl in den Befehlsfolgezähler übernommen, der von sich aus die Weiterführung des Programms veranlasst. Der Monitorstartbefehl sieht meist wie folgt aus: G C000 und bedeutet: Starte an der Adresse Hex C000.

Wie ist nun solch ein Befehl aufgebaut. Er belegt im Speicher ein, zwei oder drei Byte. Ein Byte sind 8 Bit und bilden den Inhalt eines Speicherplatzes bei einem 8 Bit Prozessor.

Das erste Byte enthält den Operationscode. Betrachten wir hierzu Tabelle 1. Hier sind alle Bitmuster, die bei einem 6510 einen Befehl darstellen, zusammengestellt. In der linken Spalte sind für diese Bitmuster leicht merkbare Ausdrücke eingegeben. Diese Schreibweise

bezeichnet man auch als Assemblerschreibweise.

Auf das Byte mit dem Operationscode können noch ein oder zwei Bytes folgen. Diese enthalten die Adresse des Speicherplatzes auf dem die Operation ausgeführt werden soll. Die Angabe dieser Adresse kann auf verschiedene Weisen, den Adressierungsarten erfolgen. Auf diese werden wir in den einzelnen Programmbeispielen eingehen.

Beispiele für Befehle

1. Laden des Akkumulators mit dem Inhalt der Speicherzelle $1000 ($ bedeutet: Folgende Zahl ist eine Hexadezimalzahl).

Assemblerschreibweise: LDA $1000

Darstellung als Bitmuster: AD 00 10

Dies ist also ein 3-Bytebefehl. Gemäß der 6510-Konvention folgt auf den Operationscode erst der niederwertige, dann der höherwertigt Adressteil.

2. Vergleiche den Akkumulator mit den Inhalt der folgenden Speicherzelle.

Assemblerschreibweise: CMP #$7F

Darstellung als Bitmuster: C9 7F

Dies ist ein 2-Bytebefehl. Das # Zeichen bedeutet unmittelbare Adressierung. Die Operation bezieht sich auf den Inhalt der auf den Operationscode folgenden Speicherzelle.

3. Schiebe den Inhalt des Akkumulators eine Stelle links

Assemblerschreibweise ASL

Darstellung als Bitmuster: OA

Dies ist ein 1-Bytebefehl, denn die Angabe einer Adresse ist hier nicht notwendig.

Stichpunkte zum Teil 1:

* Monitor
* Adresse
* Befehlsfolgezähler
* Befehl
* 1-,2-, 3-Bytebefehl

Befehle	symb. Code	Wirkung	IMM.	ABS	ABS,X	ABS,Y	Z0	Z0,X	Z0,Y	(IND,X)	(IND),Y	REL	IND	ACCU	IMPL	N	Z	C	1	D	V
			ADRESSIERUNGSARTEN																		
Transport	LDA	M → A	A9	AD	BD	B9	A5	B5		A1	B1					X	X	–	–	–	–
	LDX	M → X	A2	AE		BE	A6		B6							X	X	–	–	–	–
	LDY	M → Y	A0	AC	BC		A4	B4								X	X	–	–	–	–
	STA	A → M		8D	9D	99	85	95		81	91					–	–	–	–	–	–
	STX	X → M		8E			86		96							–	–	–	–	–	–
	STY	Y → M		8C			84	94								–	–	–	–	–	–
	TAX	A → X													AA	X	X	–	–	–	–
	TAY	A → Y													A8	X	X	–	–	–	–
	TXA	X → A													8A	X	X	–	–	–	–
	TYA	Y → A													98	X	X	–	–	–	–
	TXS	X → S													9A	–	–	–	–	–	–
	TSX	S → X													BA	X	X	–	–	–	–
	PLA	S+1 → S, Ms → A													68	X	X	–	–	–	–
	PHA	A → Ms, S–1 → S													48	–	–	–	–	–	–
	PLP	S+1 → S, Ms → P													28						
	PHP	P → Ms, S–1 → S													08	–	–	–	–	–	–
Arithmetische	ADC	A+M+C → A	69	6D	7D	79	65	75		61	71					X	X	X	–	–	X
	SBC	A–M–$\overline{C}$ → A	E9	ED	FD	F9	E5	F5		E1	F1					X	X	X	–	–	X
	INC	M+1 → M		EE	FE		E6	F6								X	X	–	–	–	–
	DEC	M–1 → M		CE	DE		C6	D6								X	X	–	–	–	–
	INX	X+1 → X													E8	X	X	–	–	–	–
	DEX	X–1 → X													CA	X	X	–	–	–	–
	INY	Y+1 → Y													C8	X	X	–	–	–	–
	DEY	Y–1 → Y													88	X	X	–	–	–	–
Logische	AND	A ∧ M → A	29	2D	3D	39	25	35		21	31					X	X	–	–	–	–
	ORA	A ∨ M → A	09	0D	1D	19	05	15		01	11					X	X	–	–	–	–
	EOR	A ⊻ M → A	49	4D	5D	59	45	55		41	51					X	X	–	–	–	–
Vergleichs-	CMP	A–M	C9	CD	DD	D9	C5	D5		C1	D1					X	X	X	–	–	–
	CPX	X–M	E0	EC			E4									X	X	X	–	–	–
	CPY	Y–M	C0	CC			C4									X	X	X	–	–	–
	BIT	A ∧ M		2C			24									7	X	–	–	–	6
Verzweigungs-	BCC	BRANCH ON C=0										90				–	–	–	–	–	–
	BCS	BRANCH ON C=1										B0				–	–	–	–	–	–
	BEQ	BRANCH ON Z=1										F0				–	–	–	–	–	–
	BNE	BRANCH ON Z=0										D0				–	–	–	–	–	–
	BMI	BRANCH ON N=1										30				–	–	–	–	–	–
	BPL	BRANCH ON N=0										10				–	–	–	–	–	–
	BVC	BRANCH ON V=0										50				–	–	–	–	–	–
	BVS	BRANCH ON V=1										70				–	–	–	–	–	–
	JMP			4C									6C			–	–	–	–	–	–
	JSR			20												–	–	–	–	–	–
Schiebe-	ASL			0E	1E		06	16						0A		X	X	X	–	–	–
	LSR			4E	5E		46	56						4A		0	X	X	–	–	–
	ROL			2E	3E		26	36						2A		X	X	X	–	–	–
	ROR			6E	7E		66	76						6A		X	X	X	–	–	–
Status-Register-	CLC	C=0													18	–	–	0	–	–	–
	CLD	D=0													D8	–	–	–	–	0	–
	CLI	I=0													58	–	–	–	0	–	–
	CLV	V=0													B8	–	–	–	–	–	0
	SEC	C=1													38	–	–	1	–	–	–
	SED	D=1													F8	–	–	–	–	1	–
	SEI	I=1													78	–	–	–	1	–	–
Versch.	NOP	NO OPER													EA	–	–	–	–	–	–
	RTS	RETURN F. SUB													60	–	–	–	–	–	–
	RTI	RETURN F. INT													40						
	BRK	BREAK													00	–	–	–	1	–	–

Abbildung 1

Notizen

Eine wichtige Routine PRTBYT

Programmieren in Maschinensprache mit den 6510 Microprozessor.

Die Beispiele in diesem Buch sind für den Commodore 64 geschrieben worden. Der Editor/Assembler "MACROFIRE" von HOFACKER wurde verwendet. Es kann jedoch jeder andere beliebige symbolische Assembler für den C-64 verwendet werden.

Die Programme verwenden einige Routinen des Commodore 64 Kernals. Zwei Beispiele sind die Ausgabe eines Zeichens auf dem Bildschirm und die Eingabe eines Zeichens von der Tastatur.

```
                 BYTE      EQU     $C023
                 CHROUT    EQU     $FFD2
                           ORG     $C000
C000:  8D23C0    PRTBYT    STA     BYTE     C023
C003:  4A                  LSR
C004:  4A                  LSR
C005:  4A                  LSR
C006:  4A                  LSR
C007:  2014C0              JSR     OUTPUT   C014
C00A:  AD23C0              LDA     BYTE     C023
C00D:  2014C0              JSR     OUTPUT   C014
C010:  AD23C0              LDA     BYTE     C023
C013:  60                  RTS
C014:  290F      OUTPUT    AND     #$0F
C016:  C90A                CMP     #$0A
C018:  18                  CLC
C019:  3002                BMI     $C01D
C01B:  6907                ADC     #$07
```

```
C01D: 6930          ADC     #$30
C01F: 4CD2FF        JMP     CHROUT
C022: 00            BRK

PHYSICAL ENDADDRESS: $C023

*** NO WARNINGS

BYTE            $C023
PRTBYT          $C000     UNUSED
CHROUT          $FFD2
OUTPUT          $C014
```

Einige Programme, beinhalten den Befehl JSR PRTBYT. Mit Hilfe dieses Unterprogramms wird der Inhalt des Akkumulators in der Form von zwei Hexadezimalzahlen ausgegeben. PRTBYT muß zusammen mit dem Programm, das PRTBYT aufruft, eingegeben werden. PRTBYT beginnt bei Adresse $C000 und wird durch den OP-Code 20 00 C0 aufgerufen. Die Beispielprogrogramme fangen bei der Adresse $C100 an. Dies ist ein geschützter Speicherraum und lässt sich gut für kleine Programme oder für die Speicherung von Daten verwenden.

Warum brauchen wir eigentlich diese PRTBYT Routine?- Die CHROUT Routine des C-64 Kernal liefert nur den ASCII Wert des Akkumulators und ist deshalb nicht sehr praktisch für die Ausgabe von Zahlen. Aus diesem Grunde verwenden wir öfters die PRTBYT Routine.

Bei der Verwendung der PRTBYT-Routine als Unterprogramm muß Zelle $C022 RTS = 60 enthalten.



Teil 2

2.1 Programmiermodell der CPU 6510

Einen Überblick über die Befehle, die ein Microprozessor ausführen kann, erhält man durch das Programmiermodell des Hardware-Bausteins. Für die 6510-CPU ist dies in Abbildung 2.1 dargestellt. Es gibt vier 8-Bit Register, den Akkumulator, das X- und Y-Register und das Statusregister. Der Programm- oder auch Befehlsfolgezähler hat 16 Bit und damit einen Adressenumfang von 0 bis 65535.

15		7 … 0
		Akkumulator
		X-Register
		Y-Register
Programmzähler MSB		Programmzähler LSB
	1	Stapelzeiger
		Prozessor-Statusregister

Abbildung 2.1
Programmiermodell 6510

Als letztes bleibt noch der Stapelzeiger. Dieser zeigt auf einen besonderen Speicherbereich, auf die Adressen $ 100 - $ 1FF, den Stapel. Zur Adressierung benutzt er nur 8 Bit, das 9. Bit ist immer 1 und wird automatisch vom Prozessor hinzugefügt.

Welche Aufgabe haben nun die einzelnen Register?

Das zentrale Register ist der Akkumulator. Alle Rechenoperationen werden über den Akkumulator ausgeführt. Wird zum Akkumulatorinhalt der Inhalt einer Speicherzelle hinzuaddiert, so ist das Ergebnis dieser Operation der neue Inhalt des Akkumulators. Auch das Umspeichern des Inhalts einer Speicherzelle erfolgt über den Akkumulator. Dieses Umspeichern kann aber auch über die Indexregister geschehen. Weiter können diese Register als Zählregister verwendet werden. Durch einen Befehl INX z.B. wird der Inahlt des X-Registers um Eins erhöht, durch DEY der Inhalt der Y-Registers um Eins erniedrigt. Ferner können mit ihnen Adressen verändert, indiziert werden. Daher auch der Name Indexregister. Von dieser Möglichkeit werden wir bei späteren Programmen häufig Gebrauch machen.

Das Statusregister zeigt den augenblicklichen Zustand eines Programmes an. In den einzelnen Bits wird das Ergebnis einer Operation festgehalten (Abbildung 2.2)

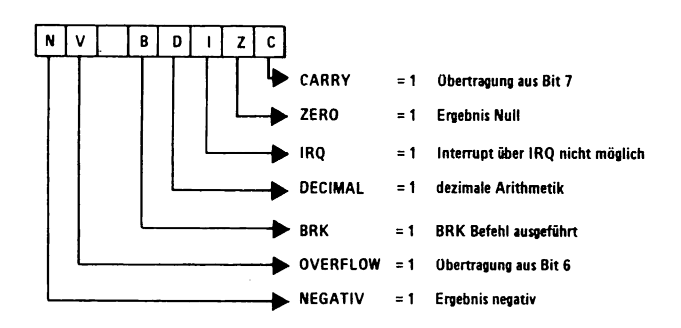


Abbildung 2.2
Belegung der Bits im Statusregister

Das Zero-Bit wird z.B. Eins, wenn der Akkumulatorinhalt gleich Null wird. Das Carry-Bit wird gesetzt, wenn bei einer Addition ein Übertrag in die nächste Stelle auftritt. In der rechten Spalte von Tabelle 1 aus dem

Im Statusregister sind die Flags

letzten Kapitel wird gezeigt, welche Operationen die einzelnen Bits im Statusregister verändern können, dabei zeigt ein X eine mögliche Änderung an. Ein LDA Befehl beeinflußt also nur das N- und Z-Bit, alle anderen nicht, während ein STA-Befehl kein Bit im Statusregister verändert.

Der Stapelzeiger zeigt, wie sein Name schon besagt, auf einen freien Speicherplatz im Stapel. Durch einen 1 Byte-Befehl PHA (Push Accumulator) wird der Inhalt des Akkumulators dort hingeschrieben, und der Stapelzeiger automatisch auf die nächste Speicherzelle gesetzt. Bei einem PLA (Pull Accumulator) wird der Stapelzeiger erst zurückgesetzt und der Inhalt dieser Zelle in den Akkumulator übernommen. Dabei ist zu beachten, daß die oberste Zelle des Stapels die Adresse $1FF ist und der Stapel zur Adresse $100 hin aufgebaut wird. Dieser Stapelspeicher hat noch eine weitere wichtige Aufgabe.

Er übernimmt bei einem Sprung in ein Unterprogramm automatisch die augenblickliche Adresse des Programmzählers. Von dort wird sie beim Rücksprung wieder in den Programmzähler übernommen.

Der Befehlsfolgezähler enthält also immer die Adresse des nächsten ausführbaren Befehls. Es wird nur durch die Sprungbefehle (JMP, JSR) verändert.

In Abbildung 2.3 sind noch einmal alle Transportbefehle für die Datenübertragung zwischen den Registern und dem Speicher gezeigt. Mann sieht, daß beim 6510 kein Befehl existiert, der den Datentransfer zwischen Speicherplatzen direkt ausführt, und ein direkter Austausch des Inhalts von X- und Y-Registern ist auch nicht möglich. Wenn man, nach der Kenntnis einer Maschinensprache, auf die Sprache eines anderen Prozessors umsteigt, so sollte man sich dessen logische Struktur genau ansehen. Daraus kann man schon im voraus feststellen, welchen Befehlsvorrat er umfaßt, und welche Wirkung die einzelnen Befehle haben.

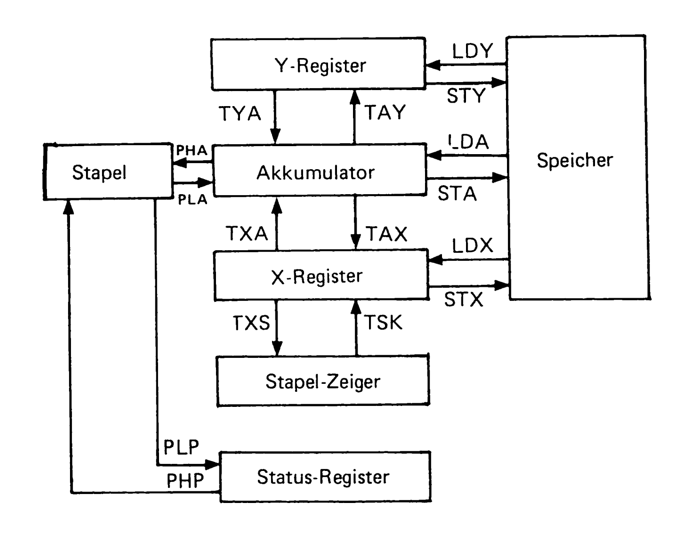


Abbildung 2.3
Datenübertragung zwischen den einzelnen Registern und dem Speicher

2.2. Ein erstes Beispiel und die Papier- und Bleistift-Methode.

Wenn man in einer höheren Programmiersprache, wie z.B. in BASIC zwei Zahlen addiert, so ist das Programm leicht hinzuschreiben.

```
10 A=5
20 B=3
30 C=A+B
40 Print C
50 END
```

=

```
OUT LNM,3
ORG $C100
LDA #$05
CLC
ADC #$03
JSR $C000
BRK
```

Will man das gleiche in Maschinensprache erreichen, so sind vor dem Programmschreiben weitere Überlegungen notwendig.

Folgende Fragen müssen zuerst beantwortet werden:
Wo sind die Zahlen gespeichert?

Sind es Festkommazahlen oder Gleitkommazahlen?
Wo soll das Programm beginnen?
Gibt es im Monitor ein Programm, das den Inhalt einer Speicherzelle ausdruckt?

Die Beantwortung dieser Fragen wollen wir bei der Umsetzung dieses BASIC-Programms in ein Maschinenprogramm vornehmen. Wir beginnen mit einer Formulierung in der Assemblerschreibweise.

Die Befehle lauten der Reihenfolge nach:

LDA #$ 05
Lade den Akkumulator mit $05. Es wird die unmittelbare Adressierung verwendet. Die Zahl $05 ist nach dem Operationscode gespeichert und ist eine Festkommazahl.

CLC
Lösche das Carrybit für den nachfolgenden Befehl: Addiere mit Übertrag (Carry)

ADC #$03
Addiere mit Übertrag $03. Er wird wieder die unmittelbare Adressierung verwendet. Das Ergebnis ist automatisch im Akkumulator.

JSR PRTBYT
PRTBYT ist ein Monitorunterprogramm, das den Inhalt des Akkumulators als zwei Hexadezimalzahlen ausgibt.

BRK
Wenn die Ausgabe beendet wird, breche hier das Programm ab.

Das Programm lautet also:

```
                    ORG $C100
C100: A905          LDA #$05
C102: 18            CLC
C103: 6903          ADC #$03
C105: 2000C0        JSR $C000
C108: 00            BRK

PHYSICAL ENDADDRESS: $C109

*** NO WARNINGS
```

Einen Überblick über dezimale und hexadezimale Adressen und Speichergrößen gibt Abbildung 2.4. Links sind die Adressen dezimal, rechts hexadezimal angegeben. Der Adressbereich von $0 bis $800 umfaßt 2K Byte Speicherplatz, der Adressbereich $C000 - $D000 4K Byte.

Dieses Programm wollen wir nun mit der "Papier- und Bleistift"-Methode in die Bitmuster eines ausführbaren Maschinenprogramms umsetzen. Dies ist zwar die unterste Methode der Assemblierung, aber dabei können weitere Kenntnisse des Programmierens erworben werden. Zuerst muß aber die Frage beantwortet werden: Wo beginnt das Programm?

Grundsaetzlich kann das Programm überall dort beginnen, wo Speicherplatz im Rechner vorhanden ist. Zwei Bereiche sollte man aber schon von vornherein nicht benutzen. Das ist einmal die "Seite NULL" oder Zero Page, die, wie wir gleich sehen werden, sehr nützlich für Hilfszellen ist und zum anderen der Stapelspeicher, da ihn auch der Prozessor benötigt. Damit sind die Speicherplätze $0 bis $1FF nicht verwendbar.

Beim C-64 bieten sich, wenn BASIC nicht verwendet wird, in der Zeropage die Adressen 02-08 hex und F1-FF hex an.

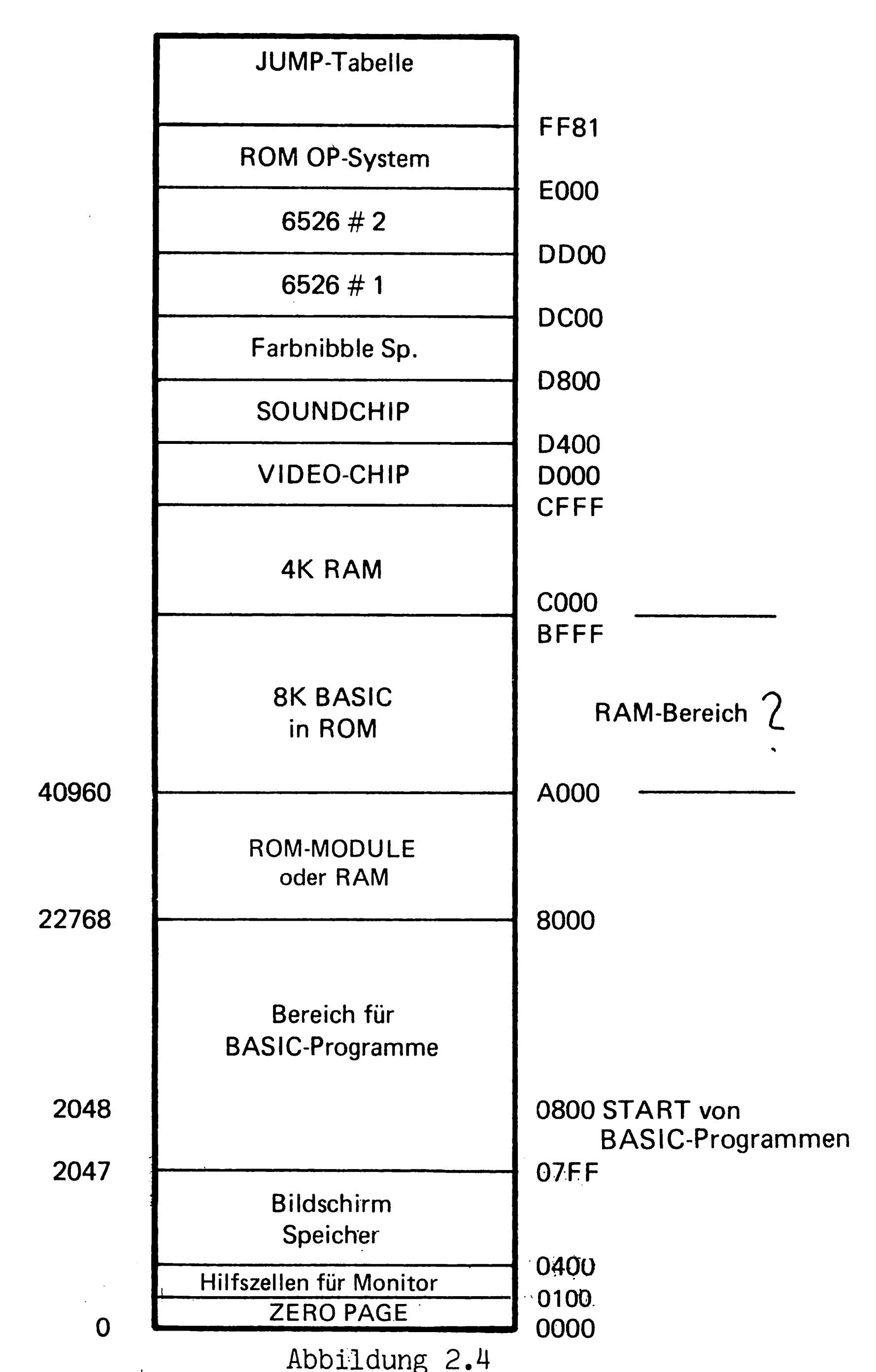


Abbildung 2.4
Dezimale und hexadezimale Adressen eines 64K Byte Speichers

Wir wissen bereits, daß ab Adresse $C000 freier Platz für unser Programm ist. Damit kann der erste Befehl umgesetzt werden. In der Tabelle 1 finden wir für LDA unmittelbar das Bitmuster A9 und damit die erste Zeile:

```
$C100        A9 05      LDA #$05
```

A9 ist der Operationscode, 05 die Zahl, die unmittelbar darauf folgt, das sind 2 Byte und die nächste Zeile beginnt in $602.

```
$C102        18         CLC
```

18 ist das Bitmuster, welches das Übertragungsbit löscht. Wir finden es in der Tabelle 1 bei den Status-Register-Befehlen. Es folgt die Zeile mit der Addition ADC (Add with Carry).
Bei der Addition wird das Übertragungsbit zum Ergebnis addiert. Deshalb wurde es in der vorangegangenen Zeile gelöscht.

```
$C103        69 03      ADC  #$03
```

Wieder wurde die unmittelbare Adressierung verwendet. Das Bitmuster 69 für ADC unmittelbar (#) findet man bei den arithmetischen Befehlen.

Nun folgt der Aufruf des Unterprogramms PRTBYT, das den Inhalt des Akkumulators auf den Bildschirm ausgibt.

Dieses Unterprogramm beginnt bei uns bei Adresse $1000. C000
Die Ausgabezeile für das Programm lautet dann, mit 20 als Bitmuster für den Befehl JSR (Jump Subroutine):

```
$C105        20 00 C0   JSR PRTBYT
```

Bei 6510 Prozessoren folgt bei Adressangabe auf das Byte mit dem Operationscode das niederwertige Adressbyte LSB (Least Significant Byte), danach das höherwertige Adressbyte MSB (Most Significant Byte). Danch wird das Programm mit

```
$C108        00          BRK
```

abgebrochen. Dabei findet bei den meisten Rechnern ein Rücksprung in den Monitor statt.Wenn nicht, muss zurück nach BASIC gesprungen werden.Dies geschieht i.a. mit RTS. Das Programm lautet somit:

```
$C100 A9 05     LDA #$05
$C102 18        CLC
$C103 69 03     ADC #$03
$C105 20 00 C0  JSR PRTBYT
$C108 00        BRK
```

Die Speicherzellen von $C100 bis $C108 haben damit folgenden Inhalt:

```
$C100: A9 05 18 69 03 20 00 C0
$C108: 00
```

Es soll im Augenblick nicht darauf eingegangen werden, wie man diese Bitmuster in den Rechner schreibt, und wie man das Programm startet. Es soll vielmehr gezeigt werden, wie sich das Programm ändert, wenn man andere Adressierungsarten verwendet.

Die Programmieraufgabe bleibt die gleiche, die beiden Zahlen 5 und 3 sind aber in 2 Zellen der Zero-Page (SQ) gespeichert.
Die Zahl 5 in Zelle $03 und die 3 in Zelle $04.
Damit erhält man:

EDITORLISTING

```
OUT LNM,3
ORG $C100
LDA $03
CLC
ADC $04
JSR $C000          ;JSR PRTBYT
BRK
```

ASSEMBLERLISTING

```
                         ORG $C100
C100: A503               LDA $03
C102: 18                 CLC
C103: 6504               ADC $04
C105: 2000C0             JSR $C000          ;JSR PRTBYT
C108: 00                 BRK
PHYSICAL ENDADDRESS: $C109

*** NO WARNINGS
```

A5 ist das Bitmuster für LDA mit dem Inhalt einer Zelle aus der Zero Page (SQ), 65 das Bitmuster für ADC mit dem Inhalt einer Zelle in der Zero Page.

Beim letzten Beispiel wollen wir annehmen, daß die beiden Zahlen irgendwo im Rechner z.B. in den Zellen $CF00 und $CF0F gespeichert sind.

Das Programm hat dann folgendes Aussehen:

EDITORLISTING

```
                         OUT LNM,3
                         ORG $C100
                         LDA $CF00
                         CLC
                         ADC $CF0F
                         JSR $C000
                         BRK
```

ASSEMBLERLISTING

```
                         ORG $C100
C100: AD00CF             LDA $CF00
C103: 18                 CLC
C104: 6D0FCF             ADC $CF0F
C107: 2000C0             JSR $C000
C10A: 00                 BRK

PHYSICAL ENDADDRESS: $C10B

*** NO WARNINGS
```

Hier ist AD das Bitmuster für den Befehl LDA mit dem Inhalt einer absoluten Adresse und 6D das Bitmuster für ADC mit dem Inhalt einer absoluten Adresse.
Das letzte Programm belegt 2 Byte mehr Speicherplatz. Benötigt man in einem Programm viele Hilfszellen, so legt man diese in die Zero-Page. Die Programme werden dadurch kürzer.

Stichpunkte zu Teil 2:

* Programmiermodell 6510
* CPU-Register
* unmittelbare Adressierung
* Adressierung der ZERO PAGE
* absolute Adressierung

Notizen



Teil 3

Im Teil 2 des Buches haben wir ein Programm in Maschinencode übersetzt, das keine Verzweigung enthielt, sondern von Anfang bis Ende geradlinig durchlaufen wurde. Im Folgenden werden wir nun Programme mit Verzweigungen betrachten.

3.1. Programmverzweigungen

Die meisten Programme werden Programmschleifen enthalten, die so lange durchlaufen werden, bis eine Bedingung erfüllt ist. Dann wird diese Schleife verlassen und das Programm fortgesetzt.

Bedingungen sind z.B., ob der Inhalt eines Registers oder einer Speicherzelle Null ist, oder ob eine Zahl in einem Register größer, gleich oder kleiner als eine Zahl in einer Speicherzelle ist. Durch Vergleiche, aber auch durch Operationen werden die Bits im Statusregister gesetzt (s. Abb.2.2). Die Verzweigungsbefehle überprüfen diese Bits und führen danach eine Verzweigung aus oder auch nicht. Das einfachste Beispiel ist eine Zeitschleife:

Der Inahlt des X-Registers wird solange um Eins erniedrigt, bis der Inhalt des Registers Null ist.

```
LDX #$0A;LADE DAS X REGISTER MIT A0
M DEX ; DEKREMENTIERE X REGISTER UM 1
BNE M ;SPRINGE ZUR MARKE M,SOLANGE
      ;NICHT NULL
BRK   ; HALTE HIER,WENN X REGISTER 0 IST
```

```
         LDX #$0A
MARKE    DEX
         BNE MARKE
         BRK
```

Dieses Programm wird nun von Hand übersetzt. Anfangsadresse ist $C100

```
C100  A2 0A      LDX #$0A
C102  CA       M DEX
C103  D0 —       BNE M
C105  00
```

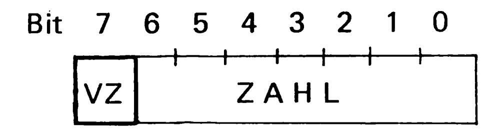
Noch nicht angegeben ist in dem 2 Byte-Befehl die Zahl der Byte, die zurückgesprungen werden. Dazu sind 2 weitere Überlegungen notwendig. Die Verzweigungsbefehle benutzen die relative Adressierung. Das bedeutet, der Befehlsfolgezähler wird um die angegebene Zahl von Bytes erhöht oder erniedrigt und das Programm an dieser Stelle fortgesetzt.

Welchen Inhalt hat nun der Befehlsfolgezähler?
Bei dem 6510 System zeigt er auf die Adresse des nächsten Befehls, in unserem Fall auf BRK in Zelle $C105. Zur Zelle $C102 muß nun um 3 Byte zurückgesprungen werden. In Zelle $C104 muß also die Hexadezimalzahl -3 stehen. Um diese zu bestimmen, müssen wir also negative Zahlen einführen.

3.2 Positive und negative Zahlen

Die Kennzeichnung von positiven und negativen Zahlen erfolgt durch Bit 7 der Zahl.

Ist dieses Bit =1,



so ist es eine negative Zahl, ist es =0, so ist es eine positive Zahl.

Damit erhält man für positive Zahlen:

0 = $00 = % 0000 0000
1 = $01 = % 0000 0001
2 = $02 = % 0000 0010
.
.
127 = $7F = % 0111 1111

Negative Zahlen werden dagegen im 2-er Complement dargestellt. Complementieren einer Zahl bedeutet das Vertauschen von 0 und 1 im Bitmuster. Bei der Bildung des 2-Complements wird zu diesem neuen Bitmuster noch eine Eins hinzuaddiert. Zur Bildung der Zahl -1 benötigen wir also das Bitmuster für die Zahl +1 = %0000 0001.

Dieses Bitmuster complementiert, ergibt % 1111 1110. Dazu 1 addiert ergibt

```
    % 1111 1110
+   % 0000 0001
---------------
    % 1111 1111 = $FF
```

Berechnen wir auf diese Weise die Zahl -3

```
          + 3 =   % 0000 0011
Complement        % 1111 1100
+ 1             + % 0000 0001
                  -----------------
                  % 1111 1101 = $FD
```

Für negative Zahlen erhält man:

```
 - 1 = $FF = % 1111 1111
 - 2 = $FE = % 1111 1110
 - 3 = $F0 = % 1111 1101
    .
    .
    .
- 128= $80 = % 1000 0000
```

Der Zahlenbereich für eine vorzeichenbehaftete 8-Bit-Zahl reicht also von -128 bis +127. Dies sind dann auch die Grenzen bei der relativen Adressierung der Verzweigungsbefehle.

Unser Programm einer Zeitschleife lautet damit:

```
                                ORG $C100
C100: A2A0                      LDX #$A0
C102: CA          MARKE         DEX
C103: D0FD                      BNE MARKE
C105: 00                        BRK

PHYSICAL ENDADDRESS: $C106

*** NO WARNINGS

MARKE               $C102
```

Für die Berechnung der Verzweigungen benutzt man am besten die Tabellen in Abbildungen 3.1 und 3.2.

LSD / MSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

Abbildung 3.1
Vorwärtsverzweigungen

LSD / MSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113
9	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97
A	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81
B	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65
C	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49
D	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
E	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
F	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Abbildung 3.2
Rückwärtsverzweigungen

Die Berechnung der Sprungweite stellt bei der Assemblierung von Hand eine der häufigsten Fehlerquellen dar.

3.3 Vergleiche

Ein Vergleich findet immer zwischen einem Register (Akkumulator, X- oder Y-Register) und einer Speicherzelle statt. Durch den Vergleich werden die Bit N (Negativ) Z (Zero) und C (Carry) gesetzt. Dies ist in Abbildung 3.3 dargestellt.

Vergleich	N	Z	C
A, X, Y < M	1*	0	0
A, X, Y = M	0	1	1
A, X, Y > M	0*	0	1

*Vergleich im 2-er Complement

Abbildung 3.3

Setzen der Bits im Statusregister durch Vergleichsbefehle
Ist der Inhalt des Akkumulators (X-,Y-Register) kleiner als der Inhalt einer Speicherzelle, so wird das Zero- und Carry-Bit im Statusregister auf Null gesetzt.

Für die beiden Bits werden die Zahlen als Werte zwischen 0 und 255 betrachtet. Das Setzen des N-Bits erfolgt als Vergleich im 2-er Complement, also für einen Wertebereich von -128 bis +127.
Dazu ein Beispiel: Der Inhalt des Akkumulators ist $FD, der Inhalt einer Speicherzelle ist $00. Bei einem Vergleich ist A > M (252 > 00) somit wird C = 1 und Z = 0. Für die verschiedenen Möglichkeiten einer Verzweigung erhält man:

Verzweigung nach MARKE, wenn ... mit

```
A<M                 BCC MARKE
A<M                 BCC MARKE
.                   BEQ MARKE
A=M                 BEQ MARKE
A>M                 BCS MARKE
A>M                 BEQ NICHT
.                       MARKE
BCS MARKE
```

Als einfaches Beispiel für Vergleiche und Verzweigungen soll folgendes Programm betrachtet werden:
Über das Tastenfeld soll ein Zeichen eingegeben werden. Dieses Zeichen wird daraufhin untersucht, ob es ein Hexadezimalzeichen, also 0 bis 9 und A bis F ist. Ist dies der Fall, so wird diese Zahl in einer Zelle EIN mit der Adresse $10 gespeichert. Wenn das Zeichen keiner Hexadezimalzahl entspricht, wird das Programm mit $00 in EIN verlassen.

Für die Eingabe verwenden wir ein Unterprogramm GETCHR, das in den meisten Monitoren enthalten ist. Das Unterprogramm fragt laufend das Tastenfeld ab, ob eine Taste gedrückt wird. Nach einem Tastendruck kehrt dieses Unterprogramm mit dem ASCII-Zeichen im Akkumulator in das Hauptprogramm zurück.

In Abbildung 3. 4 sind alle ASCII-Zeichen zusammengestellt.

LSD	MSD	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P		p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	•	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	–	=	M	]	m	}
E	1110	SO	RS	•	>	N	↑	n	~
F	1111	SI	VS	/	?	O	←	o	DEL

Abbildung 3.4
ASCII-Zeichen

Als Übung kann man dieses Programm einmal von Hand übersetzen und dabei die Sprungweiten bestimmen und überprüfen. Im Teil 4 werden wir uns mit der Verwendung von Unterprogrammen beschäftigen.

```
                        ORG     $C100
              CHRIN     EQU     $FFCF
              AUX       EQU     $FE
C100: A900              LDA     #0
C102: 85FE              STA     AUX
C104: 20CFFF            JSR     CHRIN
C107: C930              CMP     #$30
C109: 9013              BCC     L2
C10B: C947              CMP     #$47
C10D: B00F              BCS     L2
C10F: C93A              CMP     #$3A
C111: 9007              BCC     L1
C113: C941              CMP     #$41
C115: 9007              BCC     L2
C117: 18                CLC
C118: 6909              ADC     #9
C11A: 290F    L1        AND     #$0F
C11C: 85FE              STA     AUX
C11E: 00      L2        BRK

PHYSICAL ENDADDRESS: $C11F

*** NO WARNINGS

CHRIN           $FFCF
L1              $C11A
AUX             $FE
L2              $C11E
```

Abbildung 3.5
Programm ASCII-Hex

Stichpunkte zu Teil 3:

* Programmverzweigungen
* positive und negative Zahlen
* relative Adressierung
* Vergleiche



Teil 4

In diesem Kapitel wollen wir uns mit der Verwendung von Unterprogrammen beschäftigen. Unterprogramme sind selbständige Programmteile, die durch einen Unterprogrammaufruf JSR (Jump Subroutine) gestartet werden. Die Rückkehr in das Hauptprogramm erfolgt durch den Befehl RTS (Return from Subroutine).

4.1 Unterprogrammaufrufe

Als Beispiel wollen wir den Unterprogrammaufruf JSR GETCHR im Programm ASCII HEX aus dem letzten Kapitel betrachten. Die ersten Programmzeilen lauteten dort:

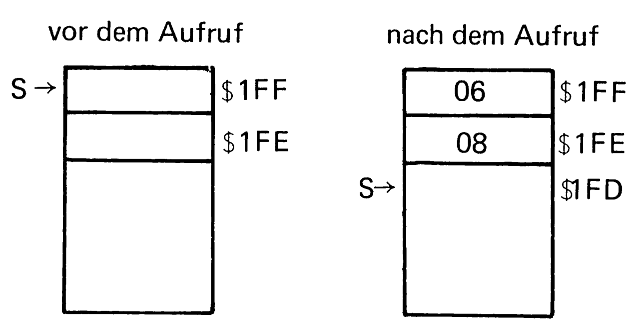
```
                          ORG $C100
C100: A900                LDA #$00
C102: 85FE                STA AUX
C104: 20CFFF              JSR CHRIN
C107: C930                CMP #$30

PHYSICAL ENDADDRESS: $C109

*** NO WARNINGS

CHRIN           $FFCF
AUX             $FE
```

In der Speicherzelle $C104 ist der Unterprogrammaufruf programmiert. Bei der Ausführung dieses Befehles wird die Adresse des nächsten Befehls (um Eins erniedrigt) in den Stapelspeicher übernommen.



Der Stapelspeicher ist bei den 6510-Systemen ein fester Speicherbereich mit dem TOS (Top of Stack) bei $1FF.
Der Stapelzeiger S zeigt immer auf die nächste freie Speicherzelle im Stapel.

Es ist moglich, aus einem Unterprogramm wiederum in ein anderes Unterprogramm zu springen. Einen solchen Programmablauf zeigt Abbildung 4.3

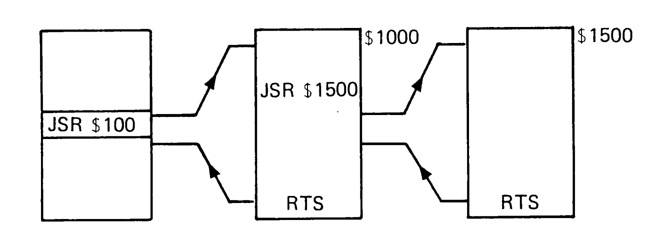


Abbildung 4.3:

Geschachtelte Unterprogrammaufrufe

Im Stapelspeicher können, wenn sonst keine anderen Daten gespeichert werden, maximal 128 Unterprogrammsprünge abgelegt werden. Dies ist eine Schachtelung, die bei normalen Programmen niemals auftritt.

4.2. Retten des Registerinhaltes

In den meisten Fällen werden in einem Unterprogramm die Registerinhalte verändert. Soll aber der Inhalt eines Registers im Hauptprogramm, nach dem Durchlaufen eines Unterprogrammes, weiter verwendet werden, so muß dieser Inhalt gerettet werden.

Dies kann im Hauptprogramm oder im Unterprogramm geschehen. Sind die im Unterprogramm benutzten Register bekannt, so brauchen nur diese im Hauptprogramm zwischengespeichert werden. Die einfachste Methode ist aber, ein im Unterprogramm benutztes Register auch dort zu retten. Der Beginn eines Unterprogramms kann dann folgendermaßen aussehen:

```
UP   PHA   ; Akkumulator in den Sta-
             pelspeicher
     TXA   ; X  → A
     PHA   ; X-Register in den Stapel-
             speicher
     TYA   ; Y  → A
     PHA   ; Y-Register in den Stapel-
             speicher
```

Vor einem Rücksprung müssen dann die Register wieder geladen werden. Das Ende dieser Unterprogramme sieht dann so aus:

```
     PLA   ; Y-Register laden
     TAY
     PLA   ; X-Register laden
     TAX
     PLA   ; Akkumulator laden
     RTS   ; Rücksprung
```

Eine andere Möglichkeit besteht darin, die Register nicht im Stapel, sondern in Hilfszellen zu speichern.

4.3 Übergabe von Daten an ein Unterprogramm

Auf die folgenden 3 Arten können Daten aus einem Hauptprogramm an ein Unterprogramm oder von einem Unterprogramm zurück an das Hauptprogramm übergeben werden.

1. Daten werden über die Register ausgetauscht. Bei den meisten Eingabeprogrammen, bei denen ein Zeichen über das Tastenfeld eingegeben wird, ist dieses nach dem Verlassen des Unterprogramms im Akkumulator.

2. Die Daten werden im Stapelspeicher abgelegt, von wo sie entweder in das Unterprogramm oder in das Hauptprogramm übernommen werden. Diese Art der Datenübernahme wird häufig bei der Verwendung von Assemblerprogrammen zusammen mit einer höheren Programmiersprache (z.B. PASCAL) angewendet.

3. Hauptprogramm und Unterprogramm benutzen einen gemeinsamen Speicherbereich für die verwendeten Daten.

Welche der 3 Möglichkeiten verwendet wird, hängt von der vorgegebenen Programmieraufgabe ab. Wird ein Programm nur von einem Porgrammierer geschrieben, so wird er die Methode wählen, die ihm am besten geeignet erscheint. Arbeiten mehrere Programmierer an der gleichen Programmieraufgabe, so muß die Art der Datenübergabe vorher festgelegt werden.

Die Verwendung von Unterprogrammen bringt folgende Vorteile:

Ein längeres Programm wird in kleinere Programmstückchen aufgeteilt. Diese sind leichter überschaubar und auch einfacher zu testen. Mit Unterprogrammen kann eine

Programmbibliothek aufgebaut werden. Auf diese kann bei der Programmierung zurückgegriffen und somit Zeit gespart werden.

4.4 Indirekter Sprung und indirekter Unterprogrammsprung

Das folgende Assemblerlisting zeigt ein Beispiel für einen indirekten Sprung.

```
SPECL:  LDA    CART       ,CHECK FOR RAM OR CART
        BNE    ENSPEC     ,GO IF NOTHING OR MAYBE RAM
        INC    CART       ,NOW DO RAM CHECK
        LDA    CART       ;IS IT ROM?
        BNE    ENSPEC     ;NO
        LDA    CARTFG     ,YES,
        AND    #$80       ,MASK OFF SPECIAL BIT
        BEQ    ENSPEC     ;BIT SET?
        JMP    (CARTAD)   ;YES, GO RUN CARTRIDGE

        CHECK FOR AMOUNT OF RAM
                                        This is an indirect jump.

        3758  F23F  AD FC BF
        3759  F242  D0 12
        3760  F244  EE FC BF
        3761  F247  AD FC BF
        3762  F24A  D0 0A
        3763  F24C  AD FD BF
        3764  F24F  29 80
        3765  F251  F0 03
        3766  F253  6C FE BF
        3767
        3768
        3769
        3770
```

Stichpunkte zu Teil 4

* Unterprogramme

* Datenübergabe
* Retten der Registerinhalte
* Indirekte Adressierung
* indirekter Sprung mit Unterprogrammsprung

TEIL 5

5.1 Die indizierte Adressierung

In diesem Teil wollen wir uns mit der indizierten Adressierung beschäftigen. Betrachten wir dazu folgendes Beispiel: Wir haben Daten (Zahlen, Buchstaben) in dem Speicherbereich $ 4000 bis $ 401F gespeichert. Diese Daten sollen nun in einen anderen Bereich, beginnend bei der Adresse $ 5000 geschrieben werden. Dafuer koennte man folgendes Programm schreiben:

```
LDA $4000 ; LADE DEN INHALT VON ZELLE $4000
STA $5000 ; SPEICHERE DIESEN NACH ZELLE $5000
LDA $4001
STA $5001
LDA $4002
STA $5002
:
:
USW BIS
LDA $401F
STA $501F
```

Abgesehen von der Schreibarbeit für dieses Programm, so benötigt man in diesem Fall 6 Programmbyte für das Umsetzen eines Datenbytes von einer Adresse zu einer anderen. Für unsere 32 Datenbyte benötigt man also ein Programm von insgesamt 32 * 6 = 192 Byte. Sollen noch mehr Daten umgesetzt werden, so würde ein Programm in dieser Form sehr lang werden. Um die Programmierung nun zu vereinfachen, benötigt mann für solche Programme zur Datenumsetzung die indizierte Adressierung mit einem der beiden Indexregister der 6510 CPU. Der Befehl LDA $4000,

X bedeutet: hole den Inhalt der Speicherzelle, deren Adresse sich aus der Summe von Adresse ($4000) und Inhalt des X-Registers ergibt. Mit LDA $4000,X wird mit X=1 der Inhalt der Speicherzelle $4001, mit X=2 der Inhalt der Speicherzelle $4002 und mit X=$1C der Inhalt der Speicherzelle $401C geholt. Anstelle des X-Registers hätte man auch das Y-Register verwenden können. Der Befehl lautet dann LDA $4000,Y.

Für unser Beispiel erhalten wir dann folgendes Programm

```
                    ORG    $C100
              FROM  EQU    $4000
              TO    EQU    $5000
              CLR   EQU    $00
C100: A200    MOVE  LDX    #CLR
C102: BD0040  M     LDA    FROM,X
C105: 9D0050        STA    TO,X
C108: E8            INX
C109: E020          CPX    #$20
C10B: D0F5          BNE    M
C10D: 00            BRK

PHYSICAL ENDADDRESS: $C10E

*** NO WARNINGS

FROM          $4000
CLR           $00
M             $C102
TO            $5000
MOVE          $C100     UNUSED
```

Abbildung 5.1
Verschieben

Das X-Register wird Null gesetzt und dann mit LDA $4000, X der Inhalt von $4000 geholt und nach $5000 geschrieben. Mit INX wird der Inhalt des X-Registers um Eins erhöht. Danach muß die Abfrage erfolgen, ob alle Daten

übertragen worden sind. Insgesamt sollen die Inhalte der Zellen $4000 bis $401F übergeben werden. Die erste Zelle, die nicht übertragen werden soll, ist die Zelle $4020. Ist der Inhalt des X-Registers nach dem Erhöhen um Eins gleich $20, so wird das Programm abgebrochen.

Ein Wort noch zu dem Kommentar in diesem Beispiel. Mit $4000 wird die Adresse einer Zelle eingegeben, mit ($4000) der Inhalt dieser Zelle.
Die beiden Indexregister X und Y sind 8-Bit-Register, somit ist eine Indizierung von 0 bis 256 möglich. Datenfelder mit maximal 256 Byte können mit dieser Adressierung übertragen werden. Sollen größere Datenfelder umgespeichert werden, so müssen andere Adressierungsarten, auf die wir gleich eingehen werden, verwendet werden.

Zuvor noch ein Programmbeispiel, bei dem die Inhalte der Zelle $4000 bis $40FF vertauscht werden. Also der Inhalt von $4000 mit $40FF, von $4001 mit $40FE, von $4002 mit $40FD usw. (Abb. 5.2)

Zuerst wird 0 nach X und FF nach Y geschrieben. Der Inhalt von $4000 wird geholt und im Stapelspeicher zwischengespeichert. Dann wird der Inhalt von $40FF nach $4000 umgespeichert und der zwischengespeicherte Wert nach $40FF geschrieben. Das Y-Register wird um Eins erniedrigt, das X-Register um Eins erhöht. Die Vertauschung ist abgeschlossen, wenn der Inhalt von X=$80 ist.

```
                         ORG     $C100
                ADDRESS  EQU     $4000
C100: A200               LDX     #$00
C102: A0FF               LDY     #$FF
C104: BD0040 M           LDA     ADDRESS,X
C107: 48                 PHA
C108: B90040             LDA     ADDRESS,Y
C10B: 9D0040             STA     ADDRESS,X
C10E: 68                 PLA
```

```
C10F: 990040        STA     ADDRESS,Y
C112: 88            DEY
C113: E8            INX
C114: E080          CPX     #$80
C116: D0EC          BNE     M
C118: 00            BRK

PHYSICAL ENDADDRESS: $C119

*** NO WARNINGS

ADDRESS             $4000
M                   $C104
```

Abbildung 5.2
Vertauschen

Bei der indizierten Adressierung wird zur Berechnung der endgültigen (effektiven) Adresse die Summe aus programmierter Adresse und Inhalt des Indexregisters gebildet. Tritt bei dieser Summenbildung ein Übertrag auf, so wird dieser berücksichtigt. Mit X=$FF wird durch den Befehl
STA $40E0,X der Akkumulatorinhalt nach $41DF geschrieben.

Der Befehlssatz der 6510 CPU besitzt noch zwei weitere Adressierungsarten, die sich aus der indirekten und der indizierten Adressierung zusammensetzen. Zur Erinnerung nochmal: Bei der indirekten Adressierung ist nicht die programmierte Adresse, sondern der Inhalt dieser Adresse die eigentliche Zieladresse.

Beispiel: JMP ($C800) bedeutet einen Sprung nach $4000, wenn dies der Inhalt der ZelleN $C800 UND $C801 ist.
$C800 enthält 00 und $C801 enthält 40.

5.2 Die indiziert-indirekte Adressierung

Beispiel: LDA ($10,X)

Die endgültige Adresse wird nun auf folgende Weise berechnet:

Zur programmierten Adresse $10 wird der Inhalt des X-Registers hinzuaddiert. Der Inhalt dieser neuen Adresse und des darauffolgenden Bytes ist die endgültige Adresse. Betrachten wir dazu folgendes Beispiel:

Der Inhalt der Zellen $0E-$15 ist:

(0E)=FF
(0F)=0F
(10)=00
(11)=11
(12)=2F
(13)=30
(14)=00
(15)=47

Der Befehl LDA ($10, X) holt mit X=0 den Inhalt der Zelle $1100, mit X=2 den Inhalt der Zelle $302F und mit X=4 den Inhalt der Zelle $4700.

Tritt bei der Berechnung der Summe aus Adresse und Registerinhalt ein Übertrag auf, so wird dieser nicht berücksichtigt.
Deshalb wird mit X=$FE der Inhalt von $0FFF geholt.

5.3 Die indirekt-indizierte Adressierung

Auch hier ist die programmierte Adresse eine Adresse aus

der Zero-Page. Als Indexregister kann das Y-Register verwendet werden.
Beispiel: STA ($10),Y

Die endgültige Adresse wird wie folgt berechnet: Zum Inhalt der programmierten Adresse in den Zellen $10, $11 wird der Inhalt des Y-Registers addiert. Dies ist dann die gültige Adresse.
Mit

($10)=3E
($11)=2F

und Y=0 wird der Inhalt des Akkumulators nach $2F3E, mit Y=$01 der Inhalt nach $2F3F gespeichert.

Beide Adressierungsarten werden in dieser vollständigen Form nicht sehr häufig in Programmen angewendet. Allerdings trifft man sie in der Form der indirekten Adressierung an. Setzt man X oder Y-Register Null, so bedeutet LDA ($10, X): Hole den Inhalt der Zelle, deren Adresse in $10 und $11 gespeichert ist. Das gleiche bedeutet auch der Befehl LDA ($10),Y mit Y=0.

Ändert man den Inhalt dieser Zellen, so kann man mit einem Befehl mit einer fest programmierten Adresse auf verschiedene Adressen zugreifen. Dies soll nun in einem kleinen Programm verwendet werden, das nicht nur 256 Byte, sondern 4K Byte von $4000 nach $5000 verschiebt. (Abb.3)

```
                            ORG     $C100
                  CLR       EQU     $00
                  LOS       EQU     $FB
                  LOD       EQU     $FD
                  HIS       EQU     $FC
                  HID       EQU     $FE
C100: A200                  LDX     #CLR
C102: 86FB                  STX     LOS
C104: 86FD                  STX     LOD
```

```
C106: A940              LDA     #$40
C108: 85FC              STA     HIS
C10A: A950              LDA     #$50
C10C: 85FE              STA     HID
C10E: A1FB     M        LDA     (LOS,X)
C110: 81FD              STA     (LOD,X)
C112: E6FB              INC     LOS
C114: E6FD              INC     LOD
C116: D0F6              BNE     M
C118: E6FC              INC     HIS
C11A: E6FE              INC     HID
C11C: A5FC              LDA     HIS
C11E: C950              CMP     #$50
C120: D0EC              BNE     M
C122: 00                BRK

PHYSICAL ENDADDRESS: $C123

*** NO WARNINGS

CLR                $00
LOD                $FD
HID                $FE
LOS                $FB
HIS                $FC
M                  $C10E
```

Abbildung 5.3
4K-Verschieben

Im Programm werden zuerst die Anfangsadressen für Start ($FB, $FC) und Ziel ($FD, $FE) festgelegt. Dann wird mit LDA ($FB,X) der Inhalt von $4000 geholt und mit STA ($FD,X) nach $5000 geschrieben. Der Zelleninhalt von $FB und $FD wird um Eins erhöht. Wenn dabei nach der Übertragung von 256 eine Page Grenze überschritten wird, muß auch der Zelleninhalt von $FC und $FE erhöht werden, bis in den Zellen $FB und $FC die Adresse der ersten, nicht zu verschiebenden Zelle ($5000) erreicht ist.

Zur Programmierübung 2 kleine Aufgaben:

1.) Programm FILL. Ein Speicherbereich, mit der Anfangsadresse in $FB und $FC und der Endadresse in $FD und $FE soll mit einer Hexzahl, die in $02 steht, gefüllt werden.

2.) Programm MOVE. Ein Datensatz mit der Anfangsadresse in $F9, $FA und der Endadresse in $FB, $FC soll zu einer Zieladresse (gespeichert in $FD,$FE) verschoben werden. Für diese Zieladresse gibt es keine Beschränkungen. Sie darf auch innerhalb des Adressbereichs des Datensatzes liegen. Dies ist bei den obigen Beispielen nicht möglich.

Stichpunkte zu Teil 5

* Indizierte Adressierung
* Indiziert-indirekte Adressierung
* Indirekte-indizierte Adressierung
* Verschieben von Daten im Speicher



TEIL 6

In diesem Kapitel wollen wir uns mit der Eingabe von Daten, also Buchstaben oder Zahlen, in den Rechner beschäftigen. Diese Daten sollen über das Tastenfeld eingegeben werden. Bei allen Rechnern mit Tastenfeld ist im Monitor ein Unterprogrmm GETCHR vorhanden, das ein, der gedrückten Taste entsprechendes Zeichen, im Akkumulator enthält. Die Codierung ist in den meisten Fällen der ASCII-Code oder ein leicht davon abweichender Code.

Dem Buchstaben A entspricht das ASCII-Zeichen $41. Diese Verschlüsselung verwendet z.B. der ATARI , der PET,der CBM sowie der VC-20 und der C-64. Beim APPLE entspricht das A dem Hex-Zeichen $C1. Hier ist bei allen normal dargestellten Zeichen Bit 8=1 gesetzt. Will man also Maschinenprogramme von einem anderen System verwenden, so muß man bei Zahlen- oder Buchstabeneingabe darauf achten, daß die richtige Zeichenverschlüsselung verwendet wird.

Beim Commodore 64 lautet die Abfrage, ob die Taste A gedrückt wurde

```
JSR GETCHR
CMP #41
```

Beim Apple dagegen

```
JSR GETCHR
CMP #C1
```

Werden in einem Programm Eingaben über das Tastenfeld verlangt, so erfolgt dies meist über sogenannte Menüprogramme. Diese aus BASIC wohlbekannte Programmiertechnik kann auch in Maschinensprache erfolgen. Hier wird man einen gespeicherten Text auf den Bildschirm ausgeben, auf die Antwort, d.h. auf das Drücken einer Taste, warten und dann in einer Schleife den Wert dieser Taste untersuchen und entsprechend verzweigen. Dies soll in einem Flußdiagramm gezeigt werden. Zuvor müssen wir aber den Begriff des Flußdiagramms und seine Elemente erläutern. In einem Flußdiagramm wird der Programmablauf durch grafische Symbole dargestellt (Abbildung 6.1).

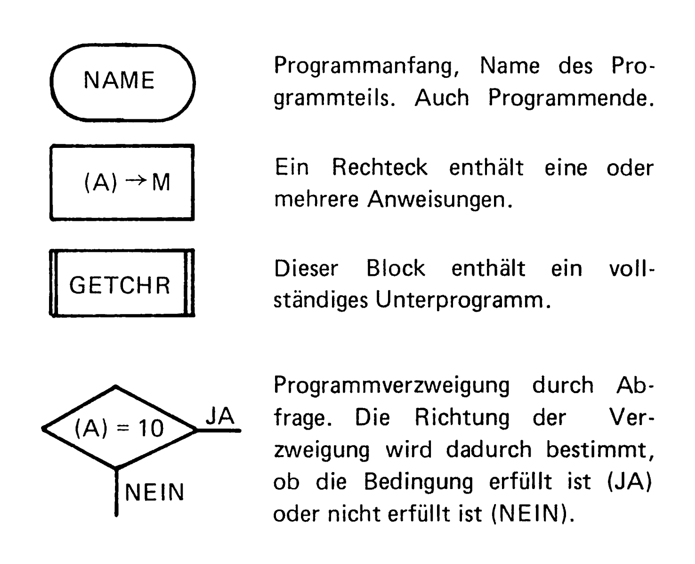


Abbildung 6.1
Elemente eines Flußdiagramms

Für unser Beispiel das Flussdiagramm in Abbildung 6.2.

Zuerst wird der erklärende Text ausgegeben, dann wird auf die Betätigung einer Taste gewartet. Wird A gedrückt, so wird Programmteil A durchlaufen, wird B gedrückt, dann wird Programmteil B erledigt und mit E das Programm beendet. War es keine dieser 3 Tasten, so wird ein akustisches Zeichen

ausgegeben und auf eine erneute Eingabe gewartet.

So einfach dies ist, zwei Bedingungen sollte so ein Menüprogramm immer enthalten. Das ist einmal eine programmierte Beendigung des Programms, also kein Abbruch durch RESET oder Rechnerausschalten, und zum zweiten, ein Abfangen von Fehlbedienungen. Dabei muß durch Ausgabe eines Zeichens, entweder akustisch oder auf den Bildschirm, angezeigt werden, daß eine Fehlbedienung vorlag.

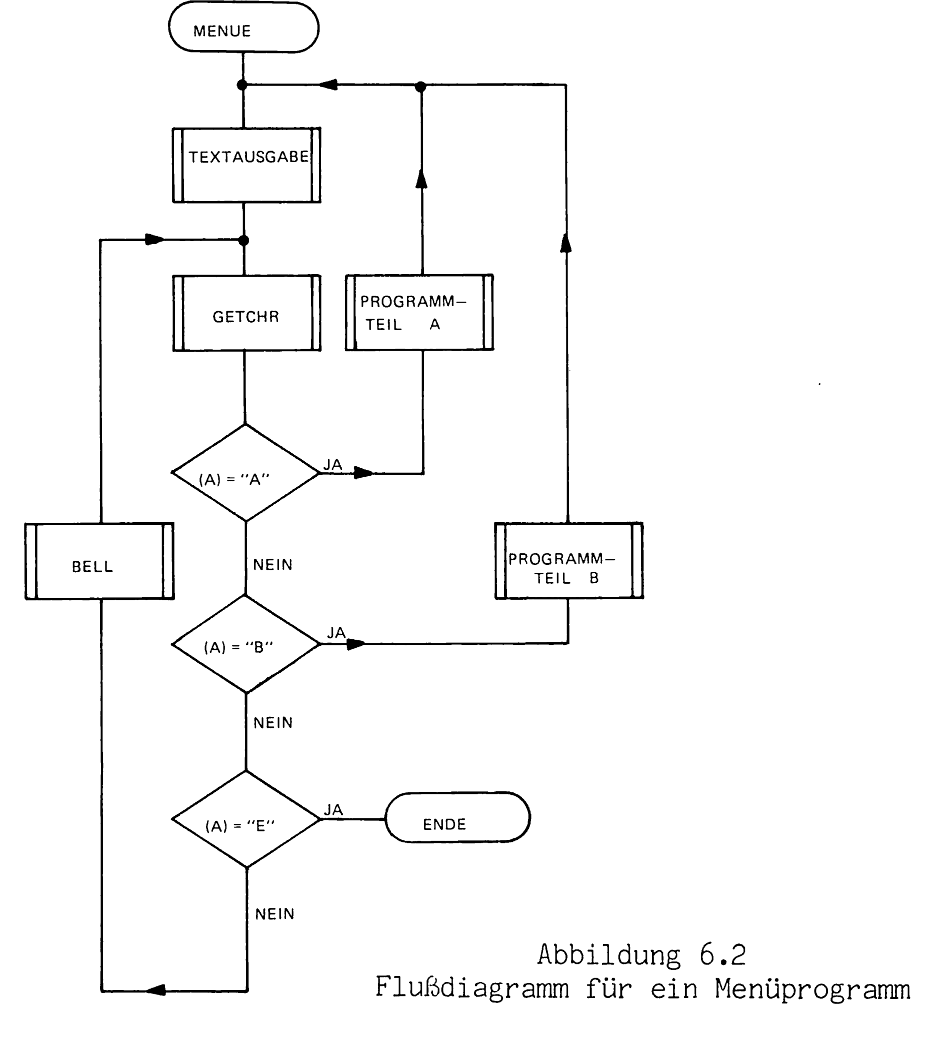


Abbildung 6.2
Flußdiagramm für ein Menüprogramm

Dazu nun das Programm. Im Programm ab Adresse $C100 wird zuerst der Bildschirm gelöscht und dann mit dem Unterprogramm TXTOUT, Text auf den Bildschirm ausgegeben. Dazu werden die ab $C140 gespeicherten Bytes nacheinander auf den Bildschirm ausgegeben.

Source-Code des Menü-Programms.

```
                       * MENU

                               ORG      $C100
                       PLOT    EQU      $FFF0
                       CHRIN   EQU      $FFCF
                       CHROUT  EQU      $FFD2
C100:  A993    MENU    LDA      #$93
C102:  20D2FF          JSR      CHROUT
C105:  202EC1  MENU1   JSR      TXTOUT
C108:  A900            LDA      #$00
C10A:  20CFFF          JSR      CHRIN
C10D:  C941            CMP      #$41
C10F:  D006            BNE      MENU2
C111:  206AC1          JSR      A
C114:  18              CLC
C115:  90EE            BCC      MENU1
C117:  C942    MENU2   CMP      #$42
C119:  D006            BNE      MENU3
C11B:  207EC1          JSR      B
C11E:  18              CLC
C11F:  90E4            BCC      MENU1
C121:  C945    MENU3   CMP      #$45
C123:  D001            BNE      MENU4
C125:  00              BRK
C126:  A907    MENU4   LDA      #$07
C128:  20D2FF          JSR      CHROUT
C12B:  18              CLC
C12C:  90D7            BCC      MENU1
C12E:  A202    TXTOUT  LDX      #$02
C130:  A003            LDY      #$03
C132:  18              CLC
C133:  20F0FF          JSR      PLOT
```

```
C136: A200              LDX     #0
C138: BD47C1  TX        LDA     TEXT,X
C13B: C99B              CMP     #$9B
C13D: F007              BEQ     TE
C13F: 20D2FF            JSR     CHROUT
C142: E8                INX
C143: 4C38C1            JMP     TX
C146: 60      TE        RTS
C147: 50524F  TEXT      ASC     "PROGRAM (A)  "
C14A: 475241
C14D: 4D2028
C150: 412920
C153: 20
C154: 50524F            ASC     "PROGRAM (B)  "
C157: 475241
C15A: 4D2028
C15D: 422920
C160: 20
C161: 454E44            ASC     "END (E) "
C164: 202845
C167: 2920
C169: 9B                DFB     $9B
C16A: A90D    A         LDA     #$0D
C16C: 20D2FF            JSR     CHROUT
C16F: A205              LDX     #5
C171: A941    AA'       LDA     #$41
C173: 86FE              STX     $FE
C175: 20D2FF            JSR     CHROUT
C178: A6FE              LDX     $FE
C17A: CA                DEX
C17B: D0F4              BNE     AA
C17D: 60                RTS
C17E: A90D    B         LDA     #$0D
C180: 20D2FF            JSR     CHROUT
C183: A205              LDX     #5
C185: A942    BB        LDA     #$42
C187: 86FE              STX     $FE
C189: 20D2FF            JSR     CHROUT
C18C: A6FE              LDX     $FE
C18E: CA                DEX
C18F: D0F4              BNE     BB
C191: 60                RTS
```

```
PHYSICAL ENDADDRESS: $C192

*** NO WARNINGS

PLOT            $FFF0
CHROUT          $FFD2
MENU1           $C105
MENU3           $C121
TXTOUT          $C12E
TE              $C146
A               $C16A
B               $C17E
CHRIN           $FFCF
MENU            $C100    UNUSED
MENU2           $C117
MENU4           $C126
TX              $C138
TEXT            $C147
AA              $C171
BB              $C185
```

Im Programm 6. 3 sind in der Assemblerschreibweise einige Befehle aufgetaucht, die keine CPU-Befehle sind. Dies sind sogenannte Pseudobefehle, die dem Assembler zusätzliche Angaben zum Programm machen. Mit diesen Pseudobefehlen werden wir uns im übernächsten Kapitel befassen.

Stichpunkte zu Teil 6:
* Texteingabe
* Flußdiagramm
* Elemente eines Flußdiagramms



TEIL 7

In Teil 6 haben wir uns mit der Eingabe von Text beschäftigt. Diesmal wollen wir uns mit der Eingabe von Zahlen befassen.

7.1 Eingabe einer Hexadezimalzahl

Zuerst einmal die Eingabe einer Hexadezimalzahl. Zur Eingabe vom Tastenfeld verwenden wir wieder das Eingabeprogramm GETCHR das uns im Akkumulator das ASCII-Zeichen übergibt, das der gerade gedrückten Taste entspricht. Im Unterprogramm PACK wird entschieden, ob das ASCII Zeichen einer Hexadezimalzahl, also den Ziffern 0 bis 9 und den Buchstaben A bis F entspricht. Trifft dies nicht zu, so verlässt das Programm mit dem ASCII-Zeichen im Akkumulator die Zahleneingabe. Das Flußdiagram von PACK zeigt Abbildung 7.1.

Der Einsprung in das Unterprogramm erfolgt mit dem ASCII-Zeichen im Akkumulator. Der erste Vergleich erfolgt mit dem Zeichen 0, der zweite Vergleich mit dem Zeichen F. Wenn das Zeichen kleiner als 0 oder größer als F ist, ist es keine Hexadezimalzahl. Zwischen dem Zeichen für 9 und dem Buchstaben A sind noch einige andere Zeichen vorhanden (siehe Teil 3, Abbildung 4.3), die auch noch durch zwei Abfragen übergangen werden müssen. Wenn das Zeichen im Akkumulator kleiner als "." ist, entspricht es den Zahlen 0 bis 9, und wenn es nicht kleiner als "A" ist, dann entspricht es den Buchstaben A bis F. In diesem Fall wird zu diesem Zeichen eine 9 hinzuaddiert. Dem Buchstaben A

entspricht das Zeichen $41. Durch eine Addition von 9 wird aus der 1 in den unteren 4 Bit eine 10. Durch 4/maliges Linksschieben wird diese Zahl in die oberen 4 Bit geschoben. Danach werden der Akkumulator, die Eingabezellen INL und INH gemeinsam mit ROL 4 mal nach links geschoben. Dabei wird immer Bit 7 über das Carrybit nach Bit 0 der folgenden Zelle geschoben.

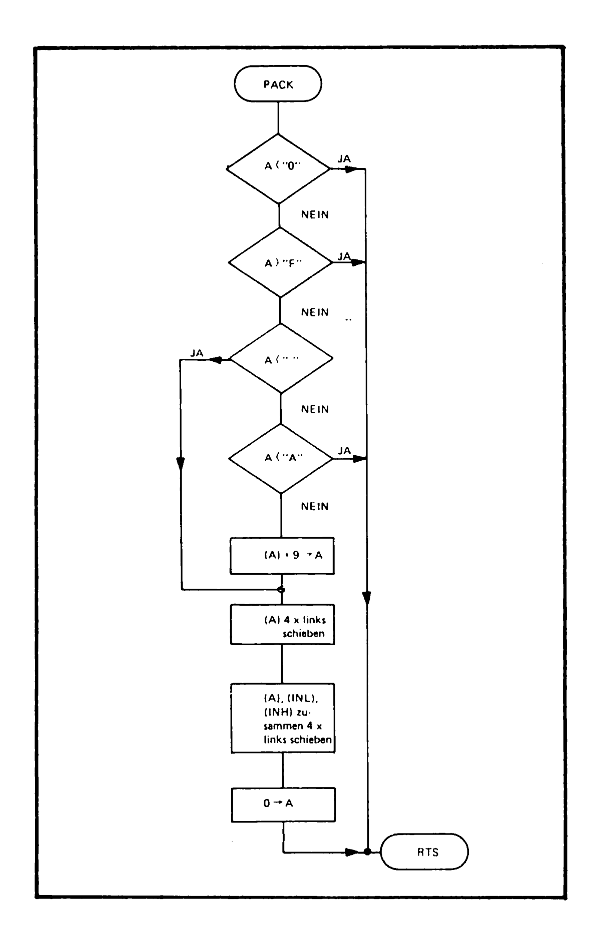


Abbildung 7.1
Flußdiagramm von PACK

Nach diesem Schiebevorgang sind die 4 Bit aus dem Akkumulator die 4 untern Bit in Zelle INL. Das Programm dazu ist in Abbildung 7.2.

```
* PACKHEX

                        ORG     $C100
               CHRIN    EQU     $FFCF
               PRTBYT   EQU     $C000
               INL      EQU     $FB
               INH      EQU     $FC
C100: C930     PACK     CMP     #$30
C102: 301F              BMI     PACKEND
C104: C946              CMP     #$46
C106: 101B              BPL     PACKEND
C108: C93A              CMP     #$3A
C10A: 3007              BMI     CALC
C10C: C941              CMP     #$41
C10E: 3013              BMI     PACKEND
C110: 18                CLC
C111: 6909              ADC     #$09
C113: 0A       CALC     ASL
C114: 0A                ASL
C115: 0A                ASL
C116: 0A                ASL
C117: A004              LDY     #$04
C119: 2A       M1       ROL
C11A: 26FB              ROL     INL
C11C: 26FC              ROL     INH
C11E: 88                DEY
C11F: D0F8              BNE     M1
C121: A900              LDA     #$00
C123: 60       PACKEND  RTS
```

7.2.Programm PACK

Das eigentliche Eingabeprogramm zeigt Abbildung 7. 3. Die beiden Eingabezellen INL und INH werden auf Null gesetzt.

Das bedeutet, daß man eine nur zweistellige Hexadezimalzahl nicht mit 004F sondern nur mit 4F einzugeben braucht. Das Zeichen wird mit CHRIN geholt und in PACK verarbeitet. Das Programm GETWD (Startadresse $C123 wird solange durchlaufen, bis ein Zeichen eingegeben wird, das keiner Hexadezimalzahl entspricht.

```
C124: A900   HEXINP   LDA     #$00
C126: 85FB            STA     INL
C128: 85FC            STA     INH
C12A: 20CFFF M2       JSR     CHRIN
C12D: 2000C1          JSR     PACK
C130: D009            BNE     INPEND
C132: A5FB            LDA     INL
C134: 290F            AND     #$0F
C136: 2000C0          JSR     PRTBYT
C139: 10EF            BPL     M2
C13B: 60     INPEND   RTS
C13C: 00              BRK

PHYSICAL ENDADDRESS: $C13D

*** NO WARNINGS

CHRIN              $FFCF
INL                $FB
PACK               $C100
M1                 $C119
HEXINP             $C124    UNUSED
INPEND             $C13B
PRTBYT             $C000
INH                $FC
CALC               $C113
PACKEND            $C123
M2                 $C12A
```

7.3 Eingabe einer 4-stelligen Hexazahl

7.2 Eingabe einer Dezimalzahl

Als nächstes soll eine Zahl als Dezimalzahl eingegeben und bei der Eingabe in eine Hexadezimalzahl gewandelt werden. Das Vorgehen ist ähnlich wie bei der Eingabe einer Hexadezimalzahl.

Es wird zuerst geprüft, ob das eingegebene Zeichen einer

Zahl von 0 bis 9 entspricht. Nun wird der bisherige Inhalt des Eingabepuffers mit 10 multipliziert und die neue Zahl hinzuaddiert.

Die 6502 CPU kennt keinen Multiplikationsbefehl. Deshalb muß die Multiplikation mit 10 anders durchgeführt werden.

Einfacher, als 10 mal hintereinander zu addieren, ist folgendes Verfahren: Ein einmaliges Linksschieben einer Zahl entspricht einer Multiplikation mit 2

```
Beispiel:      6= %00000110
                  %00001100 = 12
```

Die Zahl wird zwischengespeichert, dann 2 * nach links geschoben. Dies entspricht einer Multiplikation mit 4. Die ursprüngliche Zahl wird hinzuaddiert. Damit erhalten wir den Faktor 5. Nun brauchen wir nur noch einmal nach links schieben und die Multiplikation mit 10 ist fertig. Das Programm zeigt Abbildung 7.4.

```
                  *DEZINP

                           ORG     $C100
                  D0       EQU     $02
                  D1       EQU     $FB
                  D2       EQU     $FC
                  D3       EQU     $FD
                  D4       EQU     $FE
                  CHRIN    EQU     $FFCF
                  CHROUT   EQU     $FFD2
C100: A900        DEZINP   LDA     #$00
C102: 8502                 STA     D0
C104: 85FB                 STA     D1
C106: 20CFFF      L1       JSR     CHRIN
C109: 20D2FF               JSR     CHROUT
C10C: C930                 CMP     #$30
C10E: 303B                 BMI     L5
```

```
C110: C939              CMP     #$39
C112: 1037              BPL     L5
C114: 290F              AND     #$0F
C116: 2024C1            JSR     L3
C119: 18                CLC
C11A: 6502              ADC     D0
C11C: 8502              STA     D0
C11E: 9002              BCC     L2
C120: E6FB              INC     D1
C122: 90E2      L2      BCC     L1
C124: 85FC      L3      STA     D2
C126: A502              LDA     D0
C128: 85FD              STA     D3
C12A: A5FB              LDA     D1
C12C: 85FE              STA     D4
C12E: 2602              ROL     D0
C130: 26FB              ROL     D1
C132: 2602              ROL     D0
C134: 26FB              ROL     D1
C136: A502              LDA     D0
C138: 18                CLC
C139: 65FD              ADC     D3
C13B: 8502              STA     D0
C13D: A5FB              LDA     D1
C13F: 65FE              ADC     D4
C141: 2602              ROL     D0
C143: 26FB              ROL     D1
C145: B003              BCS     L4
C147: A5FC              LDA     D2
C149: 60                RTS
C14A: 00        L4      BRK
C14B: A99B      L5      LDA     #$9B
C14D: 20D2FF            JSR     CHROUT
C150: A5FB              LDA     D1
C152: 2000C0            JSR     $C000
C155: A502              LDA     D0
C157: 2000C0            JSR     $C000
C15A: 00                BRK

PHYSICAL ENDADDRESS: $C15B

*** NO WARNINGS
```

D0	$02	
D2	$FC	
D4	$FE	
CHROUT	$FFD2	
L1	$C106	
L3	$C124	
L5	$C14B	
D1	$FB	
D3	$FD	
CHRIN	$FFCF	
DEZINP	$C100	UNUSED
L2	$C122	
L4	$C14A	

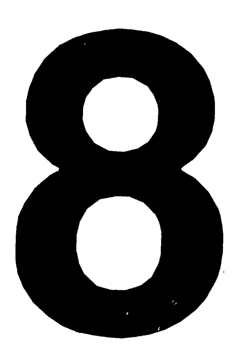
7.4 Eingabe einer Dezimalzahl

Im Programm PACK in Abbildung 7.2 entspricht das 4-malige Durchlaufen der Befehlsfolge ROL, ROL INL, ROL INH einer Multiplikation mit 16, wie sie für die Eingabe von Hexadezimalzahlen notwendig ist.

Stichpunkte zu Teil 7:

* Eingabe einer Hexadezimalzahl
* Eingabe einer Dezimalzahl
* Multiplikation mit 10

Notizen



TEIL 8

In den meisten Fällen wird man zur Programmierung in Maschinensprache einen Assembler verwenden. Ein Assembler ist ein Programm, das den mnemonischen Code in den Maschinencode übersetzt. Dieses Programm übersetzt also den Befehl LDA #$05 in die beiden Byte A9 05.

Darüber hinaus muß aber ein Assembler noch viel mehr können. Er muß z.B. immer im Programm, wenn der Name TORA auftaucht, dafür die Adresse entsprechend einsetzen. Ferner müssen Sprungziele durch Marken gekennzeichnet werden können, wie im folgenden Beispiel:

```
    LDA TORA
    BNE M1
    LDA TORB
M1  STA HFZ
.
.
.
```

Der Assembler berechnet dann automatisch die Sprungweite von Befehl BNE M1 bis zur Marke M1.

Ein Assembler besteht in den meisten Fällen aus 2 Teilen. Zuerst benötigt man einen Texteditor zum Schreiben des sogenannten Quellcodes.

Dieser Texteditor kann zeilenorientiert oder bildschirmorientiert sein. Die Zeilen müssen also entweder

mit Zeilennummern wie in BASIC oder können im freien Format eingegeben werden. Dabei verlangen die meisten Assembler, daß eine Marke für ein Sprungziel in der 1. Textstelle mit einem Buchstaben beginnt, während Befehle erst in der 2. Textstelle beginnen. Marken und Namen können in den meisten Fällen bis 6 Buchstaben lang sein.

Nach dem Schreiben des Quelltextes übersetzt der Assembler diesen Text in die Maschinensprache. Damit dies aber möglich ist, müssen ihm dazu noch weitere Befehle, sogenannte Pseudobefehle, eingegeben werden. Pseudobefehle betreffen also den Assembler, nicht aber das Programm. Leider unterscheiden sich diese Befehle für die einzelnen Assembler etwas. Die folgende Zusammenstellung von Pseudobefehlen gilt aber für die meisten Assembler.

1. ORG

Der ORG (ORIGON) legt die Anfangsadresse des Maschinencodes fest.

ORG $C100 bedeutet,

daß der Maschinencode der ersten übersetzten Zeile in der Speicherzelle mit $C100 beginnt.

Diese Adresse ist auch die Basisadresse für das folgende Programm. Auf sie beziehen sich alle absoluten Adressen. Eine ORG Anweisung muß also immer zu Beginn eines Assemblertextes gemacht werden. Sie kann aber innerhalb des Textes geändert werden.

Beispiel:

```
ORG $C100
<TEXT1>
ORG $C500
<TEXT2>
```

Der Maschinencode von TEXT1 beginnt bei Adresse $C100, der Code von TEXT2 bei Adresse $C500. Die ORG-Anweisung bewirkt ebenfalls, daß der Maschinencode auch oft als Objektcode bezeichnet, ab der angegebenen Adresse gespeichert wird.

2. OBJ
Durch den OBJ (Object)-Befehl kann nun der Maschinencode an einer anderen Stelle im Rechner gespeichert werden.

Beispiel:

```
                    ORG $3000
                    OBJ $C100
```

Das Programm wird so übersetzt, daß sich alle absoluten Adressen auf die Adresse $3000 beziehen. Der erzeugte Maschinencode wird aber ab $C100 gespeichert. Soll das Programm ausgeführt werden, so muß es erst durch einen Blocktransfer an die Adresse $3000 verschoben werden.
Beim MACROFIRE Editor Assembler sieht ein solcher Befehl wie folgt aus:

```
     ORG $3000,$C100
          :     :
          :     :
          :
          :     physikalische Adresse
          :
       logische Adresse
```

3. END
Die END-Anweisung zeigt dem Assembler an, daß hier der zu übersetzende Text aufhört.

4. EQU
Mit der EQU (Equate) Anweisung wird einer Adresse ein symbolischer Name zugeordnet.

Beispiel: TORA EQU $COCO

Die symbolische Bezeichnung TORA entspricht der Adresse $COCO.
In diesem Fall wird der Name TORA als Marke behandelt. Das Wort TORA muß also an der ersten Textstelle beginnen. Einige Assembler benötigen für die Zuweisung einer Adresse in der ZERO-Page einen gesonderten Befehl EPZ.

```
HFZ EPZ $10
```

Der symbolische Name HFZ entspricht der Adresse $10 in der Zero-Page.

5.HEX

Mit der HEX-Anweisung können Daten in Form von Hexadezimalzahlen innerhalb des Programms gespeichert werden.

Beispiel:

```
DATA HEX 00AFFC05
```

Die Zahlenfolge 00 AF FC 05 wird in 4 aufeinanderfolgende Bytes ab der symbolischen Adresse DATA gespeichert.

6. ASC

Soll in einem Programm Text als Daten abgelegt werden, so kann hierfür die ASC-Anweisung verwendet werden.

Beispiel: TEXT ASC "DIES IST EIN TEXT"

An der symbolischen Adresse TEXT wird der in Anführungsstrichen stehende Text als ASCII-Zeichen

verschlüsselt gespeichert.

Einige Assembler verwenden hierfür auch den Befehl BYT.

```
BYT 0045AF   entspricht HEX 0045AF und
BYT "TEXT"   entspricht ASC "TEXT"
```

Auf weitere Pseudobefehle soll hier nicht eingegangen werden. Sie sind den jeweiligen Handbüchern für die Assembler zu entnehmen. Vielleicht sollten wir noch den Pseudoopcode OUT besprechen, der immer wieder in unseren Beispiel-Listings vorkommt. Die wichtigsten OUT Befehle sind :

1. Assemblieren auf den Bildschirm im MACROFIRE
OUT LNM,3

2. Assemblieren auf den Commodore Drucker VC 1525 (Seikosha 100 VC)

Man schreibt in das Quelltextlisting oben OUT LNMC,4,0 <RETURN> Haben Sie einen EPSON an Device #4 ueber den User Port oder über die serielle Schnittstelle angeschlossen, geben Sie einfach nur OUT LNM, 4, 0 <RETURN>. Siehe hierzu jedoch genauere Informationen im Handbuch vom MACROFIRE Editor/Assembler.

In der Assemblerschreibweise können im Adressteil auch Adressrechnungen durchgeführt werden.

Im Programm ist folgende Pseudoanweisung programmiert:

```
DATA HEX 00AFFC05
```

Dann wird mit LDA DATA die Hexzahl 00 und mit LDA DATA+2 die Hexzahl FC in den Akkumulator übernommen.
Verwendet man die Adressrechnung bei relativen Sprüngen, so muß man aufpassen, denn durch die Angabe

BNE*+2

wird um 2 Byte und nicht um 2 Assemblerzeilen weitergesprungen. Der * ist bei einigen Assemblern ebenfalls ein Pseudobefehl, hier besser eine Pseudoadressangabe, denn er gibt den augenblicklichen Stand des Befehlsfolgezählers an.

Beispiel:

```
LDA HFZ
BNE *+2
LDA #$FF
STA HFZ
```

Wenn der Inhalt von HFZ nicht Null ist, wird der Befehl LDA #$FF übersprungen.
Einige Assembler lassen in der Adressrechnung die vier Grundrechnungsarten zu, in den meisten Fällen wird man mit Addition und Subtraktion auskommen. Als letztes soll noch auf eine kleine Besonderheit bei dieser Schreibweise eingegangen werden.

Ist im Programm

H EQU $2F   programmiert,

so bedeutet LDA H, hole den Inhalt der Zelle $2F, aber LDA #H bedeutet, hole $2F in den Akkumulator.

Stichpunkte zu Teil 8:

* Pseudobefehle
* Adressrechnungen



TEIL 9

In diesem vorletzten Teil über Programmieren in Maschinensprache 6510 wollen wir uns noch mit einigen kleinen Tricks und Besonderheiten befassen.

Bei speziellen Programmen möchte man vom Programm aus feststellen, wo sich dieses überhaupt im Speicher befindet. Dies wird bei Programmen benötigt, die absolute Adressen enthalten und trotzdem überall im Speicher lauffähig sein sollen. Beim APPLE II wird so ein Trick angewendet, um herauszufinden, in welchen Einschub eine Peripheriekarte gesteckt wurde. Da es keinen Befehl gibt, der ein Lesen des Befehlsfolgezählers ermöglicht, behilft man sich mit folgendem Trick: Vom Programm wird ein JSR-Aufruf direkt auf ein RTS, zum Beispiel im Monitor, ausgeführt. Dabei wird die augenblickliche Adresse auf den Stapel geschrieben. Man muß allerdings dabei beachten, daß das niedrigere Adressbyte um Eins erniedrigt ist. Abbildung 1 zeigt den Stapelzeiger vor, während und nach einem Unterprogrammsprung.

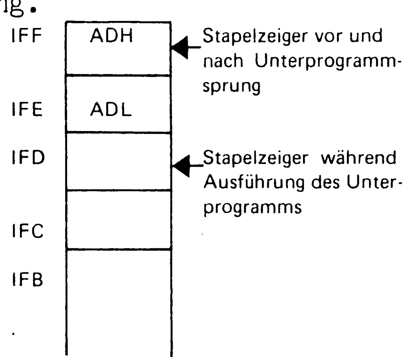


Abbildung 1:
Stapelzeiger bei Ausführung eines JSR

Nach Rückkehr ins Hauptprogramm kann der Wert des Stapelzeigers mit TSX in das X-Register übernommen werden und so auf die Adresse ADH zugegriffen werden, wie im Programm in Abbildung 2.

Wenn man anders programmieren will, kann man einen indirekten Sprung als JMP (ADR) auch folgendermaßen programmieren. Angenommen, der indirekte Sprung soll nach $2010 ausgeführt werden, dann geschieht dies mit folgedem Programm:

```
LDA #$20
PHA
LDA #$0F
PHA
RTS
```

Das ist eine Programmfolge, die häufig im ATARI Betriebssystem zu finden ist. Die übliche Art, einen indirekten Sprung zu programmieren ist

```
LDA #$10
STA ADR
LDA #$20
STA ADR + 1
JMP (ADR)
```

Verwendet man für die Speicherzelle ADR eine Adresse aus der Zero-Page, dann ist das erste Programm um 4 Byte, und wenn man eine beliebige Adresse verwendet, um 6 Byte kürzer. Dazu noch ein Vergleich der Ausführungszeiten.

LDA #$20	2	LDA #$10	2	2
PHA	3	STA ADR	3	4
LDA #$0F	2	LDA #$20	2	2
PHA	3	STA ADR+I	3	4
RTS	6	JMP (ADR)	5	5
	16		15	16

Die Zahlenangaben hinter den Befehlen sind die Zahl von Maschinenzyklen, die für die Ausführung dieses Befehls benötigt werden. Die erste Reihe hinter dem 2. Programm gilt, wenn ADR eine Zelle in der Zero-Page, die 2. wenn sie eine beliebige Adresse ist. Diese Zahl von Maschinenzyklen kann man aus den 6502 Programmiertabellen entnehmen.

Im allgemeinen wird man sich über Programmlaufzeiten in Maschinensprache keine Gedanken machen, es sei denn, man hat Programme, in denen eine Schleife sehr oft durchlaufen wird.

Dazu noch der Vergleich von zwei Programmteilen aus einem Programm zum Umspeichern von Daten. Bei beiden Programmen ist der vorangehende und der folgende Teil weggelassen, da dieser bei beiden gleich ist.

1. Programm

```
     LDA   (WOHER,X)     6
     STA   (WOHIN,X)     6
     INC   WOHER         5
     BNE   M             2 (+ 1)
     INC   WOHER +1      5
M    INC   WOHIN         5
     BNE   MI            2 (+1)
     INC   WOHIN + 1     5
MI                     ----
                         36
```

Das Programm benötigt also, wenn die Sprünge nicht ausgeführt werden, 36 Zyklen. Bei Sprungbefehlen erhöht sich die Ausführungszeit um 1 Zyklus, wenn dieser Befehl ausgeführt wird.

2. Programm

```
MEM   LDA  WOHER    4
      STA  WOHIN    4
      INC  MEM +1   5
      BNE  M        2 (+1)
      INC  MEM +2   5
M     INC  MEM+4    5
      BNE  MI       2 (+1)
      INC  MEM +5   5
                   -----
MI                 32
```

Das zweite Programm benötigt 4 Zyklen weniger, allerdings ist es ein Programm, das seinen Code verändert, denn die Zelle MEM+1 ist das niedere Adressbyte des gleichen Befehls.
Dieses Programmstück muß also im RAM stehen, es ist nicht mehr ROM-fähig.
Die Einsparung von 4 Zyklen gleich 4 us bei einer Taktfrequenz von 1 MHz ist nicht viel, kann aber bei der Übertragung größerer Datenmengen durchaus ins Gewicht fallen.

Wird in einem Unterprogramm direkt vor dem Rücksprung mit RTS ein anderes Unterprogramm aufgerufen, so kann man dabei 7 Maschinenzyklen einsparen, wenn dieser Aufruf mit JSR durch einen JMP-Befehl ersetzt wird.
Also anstatt

```
JSR NACH
RTS
```

verwenden Sie:

```
JMP NACH
```

Das RTS im Unterprogramm WOHIN führt an die gleiche Stelle wie das RTS nach dem JSR WOHIN.

Der 6510 Prozessor kennt zwar einen indirekten Sprung

JMP (ADR) aber keinen indirekten Unterprogrammaufruf JSR (ADR).
So etwas benötigt man, wenn man an einer Stelle im Program je nach Programmablauf verschiedene Unterprogramme aufrufen will, ähnlich wie bei der ON. . . GOTO-Anweisung in BASIC.

Ist dieses Programm in RAM, so wird man selbst modifizierenden Code verwenden, also die Adresse im JSR durch das Programm ändern. Dies gilt nicht mehr, wenn es ein Programm in ROM ist.

Dann kann man folgenden Trick anwenden. Irgendwo im Speicher steht die Befehlsfolge
JMPI JMP (ADR) 6CXX XX
An die Stelle von XX XX wird die Adresse des auszuführenden Unterprogramms geschrieben, und dies mit

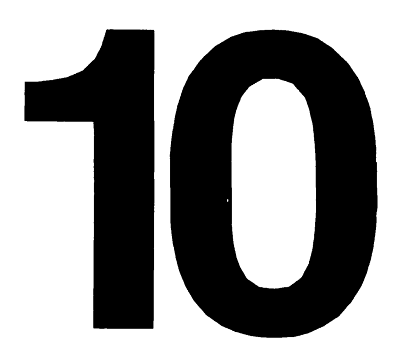
JSR JMPI

aufgerufen. Das RTS im Unterprogramm führt dann auf den Befehl nach dem JSR JMPI zurück.

Stichpunkte zu Teil 9:

* Lesen des Befehlsfolgezählers
* Zeitprobleme
* indirekte Unterprogrammaufrufe

Notizen



Einige Beispiele in Maschinensprache

Die folgenden kurzen Programme sind Beispiele für die Programmierung in Assembler-Sprache. Bei den ersten drei Programmen haben wir zusätzlich das entsprechende BASIC-Programm gelistet.

Das erste Programm füllt eine Zeile am Anfang des Bildschirms mit dem Buchstaben "C".

Das zweite Program füllt den Bildschirm mit dem Buchstaben, der eingegeben wird.

Das dritte Programm erlaubt es Ihnen, die Farben auf dem Bildschirm zu ändern.Wenn Sie die Taste "B" drücken, ändert sich die Hintergrundfarbe.Wenn Sie "S" drücken, wird die Farbe des Bildschirmes verändert. R liefert wieder die Ausgangsfarben.

Eine Zeile mit dem Buchstaben "C"

1. Programm: Reihe C in BASIC

```
100 REM ROW OF CHARACTER C
110 PRINT""
120 FORX=1TO40
130 PRINT"C";
140 NEXTX
150 END
```

Reihe C in Maschinensprache

```
                    *CROW

                                ORG       $C100
                    CHROUT      EQU       $FFD2
                    CHRIN       EQU       $FFCF
                    AUX         EPZ       $FB
C100: 4C08C1                    JMP       START
C103: A993          CLEAR       LDA       #$93
C105: 4CD2FF                    JMP       CHROUT
C108: 2003C1        START       JSR       CLEAR
C10B: A228                      LDX       #40
C10D: 86FB          S1          STX       AUX
C10F: A943                      LDA       'C'
C111: 20D2FF                    JSR       CHROUT
C114: A6FB                      LDX       AUX
C116: CA                        DEX
C117: D0F4                      BNE       S1
C119: 20CFFF                    JSR       CHRIN
C11C: 00                        BRK

PHYSICAL ENDADDRESS: $C11D

*** NO WARNINGS

CHROUT            $FFD2
AUX               $FB
START             $C108
CHRIN             $FFCF
CLEAR             $C103
S1                $C10D
```

2. Programm

```
100 REM SCREEN FULL OF CHARACTER
110 PRINT""
120 GET A$:IF A$=""THEN 120
130 FOR Y=1 TO 25
140 FOR X=1 TO 40
150 PRINT A$;
160 NEXT X
170 NEXT Y
180 GOTO 180
```

```
                    *SCREENCH

                                  ORG      $C100
                    CHROUT        EQU      $FFD2
                    CHRIN         EQU      $FFCF
                    AUX1          EPZ      $FB
                    AUX2          EPZ      $FE
C100: 4C08C1                      JMP      START
C103: A993          CLEAR         LDA      #$93
C105: 4CD2FF                      JMP      CHROUT
C108: 20CFFF        START         JSR      CHRIN
C10B: 85FE                        STA      AUX2
C10D: 2003C1                      JSR      CLEAR
C110: A019                        LDY      #25
C112: A228          S0            LDX      #40
C114: 86FB          S1            STX      AUX1
C116: A5FE                        LDA      AUX2
C118: 20D2FF                      JSR      CHROUT
C11B: A6FB                        LDX      AUX1
C11D: CA                          DEX
C11E: D0F4                        BNE      S1
C120: 88                          DEY
C121: D0EF                        BNE      S0
C123: 20CFFF                      JSR      CHRIN
C126: 00                          BRK

PHYSICAL ENDADDRESS: $C127

*** NO WARNINGS
```

```
CHROUT        $FFD2
AUX1          $FB
CLEAR         $C103
S0            $C112
CHRIN         $FFCF
AUX2          $FE
START         $C108
S1            $C114
```

3. Programm

```
100 REM BORDER AND SCREEN COLOR
110 BO=53280
120 SC=53281
130 A=PEEK(BO)
140 B=PEEK(SC)
150 GET A$:IF A$=""THEN 150
160 IFA$<>"B"THEN200
170 IF(PEEK(BO)AND15)=15THENPOKEBO,PEEK(BO)AND240:GOTO150
180 POKEBO,PEEK(BO)+1
190 GOTO150
200 IF A$<>"S"THEN 240
210 IF(PEEK(SC)AND15)=15THENPOKESC,PEEK(SC)AND240:GOTO150
220 POKESC,PEEK(SC)+1
230 GOTO150
240 IFA$<>"R"THEN150
250 POKEBO,A
260 POKESC,B
270 END
```

Hier das dazugehörige Maschinencode-Programm Decoder.

```
                    *SETCOL

                           ORG      $C100
                    CHRIN  EQU      $FFCF
                    COLOR  EQU      $D020
                    AUX    EPZ      $FB
C100: 4C0EC1               JMP      START
C103: AD20D0 COLSAV        LDA      COLOR
C106: 85FB                 STA      AUX
```

```
C108: AD21D0            LDA     COLOR+1
C10B: 85FC              STA     AUX+1
C10D: 60                RTS
C10E: 2003C1  START     JSR     COLSAV
C111: 20CFFF  S0        JSR     CHRIN
C114: C942              CMP     'B'
C116: D003              BNE     S1
C118: 202CC1            JSR     BCOLOR
C11B: C953    S1        CMP     'S'
C11D: D003              BNE     S2
C11F: 2048C1            JSR     SCOLOR
C122: C952    S2        CMP     'R'
C124: D003              BNE     S3
C126: 4C64C1            JMP     RCOLOR
C129: 18      S3        CLC
C12A: 90E5              BCC     S0
C12C: AD20D0  BCOLOR    LDA     COLOR
C12F: 290F              AND     #$0F
C131: C90F              CMP     #$0F
C133: D009              BNE     B1
C135: AD20D0            LDA     COLOR
C138: 29F0              AND     #$F0
C13A: 8D20D0            STA     COLOR
C13D: 60                RTS
C13E: AD20D0  B1        LDA     COLOR
C141: 18                CLC
C142: 6901              ADC     #1
C144: 8D20D0            STA     COLOR
C147: 60                RTS
C148: AD21D0  SCOLOR    LDA     COLOR+1
C14B: 290F              AND     #$0F
C14D: C90F              CMP     #$0F
C14F: D009              BNE     SC1
C151: AD21D0            LDA     COLOR+1
C154: 29F0              AND     #$F0
C156: 8D21D0            STA     COLOR+1
C159: 60                RTS
C15A: AD21D0  SC1       LDA     COLOR+1
C15D: 18                CLC
C15E: 6901              ADC     #1
C160: 8D21D0            STA     COLOR+1
C163: 60                RTS
C164: A5FB    RCOLOR    LDA     AUX
```

```
C166: 8D20D0         STA      COLOR
C169: A5FC           LDA      AUX+1
C16B: 8D21D0         STA      COLOR+1
C16E: 00             BRK

PHYSICAL ENDADDRESS: $C16F

*** NO WARNINGS

CHRIN          $FFCF
AUX            $FB
START          $C10E
S1             $C11B
S3             $C129
B1             $C13E
SC1            $C15A
COLOR          $D020
COLSAV         $C103
S0             $C111
S2             $C122
BCOLOR         $C12C
SCOLOR         $C148
RCOLOR         $C164
```

Relocator

Relocator für Commodore 64

Dieser Relocator für den Commodore-64 wurde mit Hilfe des Editor/Assemblers "MACROFIRE" eingegeben Sie können damit Maschinenprogramme im Speicher verschieben. Sie können zwischen echtem Relokatieren und Blocktransfer ohne Adressenumrechnung (absolute) wählen. Wenn Sie z. B. ein Programm, welches von $4000 bis $4100 im Speicher steht, nach $5000 relocatieren wollen und es befindet sich der Befehl JMP $4020 in diesem Programm, so wird dieser Befehl in JMP $5020 vom Relocator verändert.

Bevor Sie das Programm an Adresse $C100 starten können, müssen verschiedene Adressen festgelegt werden.

Auch muss der verfügbare Speicherbereich durch Angabe einer Anfangs-und Endadresse angegeben werden.

Ganz besonders wichtig beim Relokatieren ist das Aufsuchen von Text Teilen sowie von Tabellen(Sprungtabellen). Der Relokator könnte diesen Text als Opcode versuchen zu interpretieren und zu verändern.

Im nachfolgenden geben wir Ihnen eine Tabelle, die alle Adressen in der Zeropage enthält, die entsprechend vorbereitet werden müssen.

Speicherzelle	Label	Bemerkungen
7C	RFLAG	0=Relokatieren 1=Blocktransfer
7D LSB 7E MSM	TEST1	Unterste Adresse des verfügbaren Speicherbereiches.
7F LSB 80 MSB	TEST2	Oberste Adresse des verfügbaren Speicherbereiches.
81 LSB 82 MSB	START	Startadresse des zu verschiebenden Programmes.
83 LSB 84 MSB	STOP	Endadresse des zu verschiebenden Programmes.
85 LSB 86 MSB	BEG	Zieladresse=Neue Anfangsadresse

Bevor Sie den Relocator bei Adresse $C100 starten, müssen Sie die Anfangs-, End- und Zieladresse des zu verschiebenden Programms eingeben.

Bitte überprüfen Sie Ihr Programm, ob es Tabellen und Text beinhaltet, weil der Relocator diese Daten als OP-Code lesen und möglicherweise einige Bytes aendern würde.

Das folgende Listing zeigt nun den Assembler-Text für den MACROFIRE Editor/Assembler.

```
*RELOC

         ORG     $C100
RFLAG    EQU     $7C
TEST1    EQU     $7D
TEST2    EQU     $7F
START    EQU     $81
STOP     EQU     $83
BEG      EQU     $85
OPTR     EQU     $87
TEMP2    EQU     $89
NPTR     EQU     $8B
```

```
                TEMP1   EQU     $8D
C100: A205      BEGIN   LDX     #$5
C102: B581      S10     LDA     START,X
C104: 9587              STA     OPTR,X
C106: CA                DEX
C107: 10F9              BPL     S10
C109: E8                INX
C10A: A57C      MOVE    LDA     RFLAG
C10C: F006              BEQ     MO1
C10E: 204EC1            JSR     MOV1
C111: 4C5FC1            JMP     DONE
C114: A187      MO1     LDA     (OPTR,X)
C116: A8                TAY
C117: D006              BNE     MO2
C119: 2052C1            JSR     SKIP
C11C: 4C5FC1            JMP     DONE
C11F: 204EC1    MO2     JSR     MOV1
C122: C920              CMP     #$20
C124: D003              BNE     BYTE1
C126: 4C79C1            JMP     BYTE3
C129: 98        BYTE1   TYA
C12A: 299F              AND     #$9F
C12C: F031              BEQ     DONE
C12E: 98                TYA
C12F: 291D              AND     #$1D
C131: C908              CMP     #$8
C133: F02A              BEQ     DONE
C135: C918              CMP     #$18
C137: F026              BEQ     DONE
C139: 98                TYA
C13A: 291C              AND     #$1C
C13C: C91C              CMP     #$1C
C13E: F039              BEQ     BYTE3
C140: C918              CMP     #$18
C142: F035              BEQ     BYTE3
C144: C90C              CMP     #$0C
C146: F031              BEQ     BYTE3
C148: 204EC1            JSR     MOV1
C14B: 4C5FC1            JMP     DONE
C14E: A187      MOV1    LDA     (OPTR,X)
C150: 818B              STA     (NPTR,X)
C152: 20D9C1    SKIP    JSR     IOPTR
C155: 20E0C1            JSR     INPTR
```

```
C158: 60                RTS
C159: 204EC1  MOV2      JSR     MOV1
C15C: 204EC1            JSR     MOV1
C15F: A587    DONE      LDA     OPTR
C161: 858D              STA     TEMP1
C163: A588              LDA     OPTR+1
C165: 858E              STA     TEMP1+1
C167: A583              LDA     STOP
C169: 8589              STA     TEMP2
C16B: A584              LDA     STOP+1
C16D: 858A              STA     TEMP2+1
C16F: 20CEC1            JSR     TEST
C172: 9096              BCC     MOVE
C174: F094              BEQ     MOVE
C176: 00                BRK
C177: EA                NOP
C178: EA                NOP
C179: A187    BYTE3     LDA     (OPTR,X)
C17B: 858D              STA     TEMP1
C17D: 20D9C1            JSR     IOPTR
C180: A187              LDA     (OPTR,X)
C182: 858E              STA     TEMP1+1
C184: 20E7C1            JSR     DOPTR
C187: A57D              LDA     TEST1
C189: 8589              STA     TEMP2
C18B: A57E              LDA     TEST1+1
C18D: 858A              STA     TEMP2+1
C18F: 20CEC1            JSR     TEST
C192: F002              BEQ     B10
C194: 90C3              BCC     MOV2
C196: A57F    B10       LDA     TEST2
C198: 8589              STA     TEMP2
C19A: A580              LDA     TEST2+1
C19C: 858A              STA     TEMP2+1
C19E: 20CEC1            JSR     TEST
C1A1: F002              BEQ     B20
C1A3: B0B4              BCS     MOV2
C1A5: 38      B20       SEC
C1A6: A187              LDA     (OPTR,X)
C1A8: E581              SBC     START
C1AA: 8589              STA     TEMP2
C1AC: 20D9C1            JSR     IOPTR
C1AF: A187              LDA     (OPTR,X)
```

```
C1B1: E582              SBC     START+1
C1B3: 858A              STA     TEMP2+1
C1B5: 20D9C1            JSR     IOPTR
C1B8: 18                CLC
C1B9: A589              LDA     TEMP2
C1BB: 6585              ADC     BEG
C1BD: 818B              STA     (NPTR,X)
C1BF: 20E0C1            JSR     INPTR
C1C2: A58A              LDA     TEMP2+1
C1C4: 6586              ADC     BEG+1
C1C6: 818B              STA     (NPTR,X)
C1C8: 20E0C1            JSR     INPTR
C1CB: 4C5FC1            JMP     DONE
C1CE: A58E     TEST     LDA     TEMP1+1
C1D0: C58A              CMP     TEMP2+1
C1D2: D004              BNE     T10
C1D4: A58D              LDA     TEMP1
C1D6: C589              CMP     TEMP2
C1D8: 60       T10      RTS
C1D9: E687     IOPTR    INC     OPTR
C1DB: D002              BNE     INC10
C1DD: E688              INC     OPTR+1
C1DF: 60       INC10    RTS
C1E0: E68B     INPTR    INC     NPTR
C1E2: D002              BNE     INC20
C1E4: E68C              INC     NPTR+1
C1E6: 60       INC20    RTS
C1E7: C687     DOPTR    DEC     OPTR
C1E9: A587              LDA     OPTR
C1EB: C9FF              CMP     #$FF
C1ED: D002              BNE     D10
C1EF: C688              DEC     OPTR+1
C1F1: 60       D10      RTS

PHYSICAL ENDADDRESS: $C1F2

*** NO WARNINGS

RFLAG           $7C
TEST2           $7F
STOP            $83
OPTR            $87
NPTR            $8B
```

```
BEGIN         $C100    UNUSED
MOVE          $C10A
MO2           $C11F
MOV1          $C14E
MOV2          $C159
BYTE3         $C179
B20           $C1A5
T10           $C1D8
INC10         $C1DF
INC20         $C1E6
D10           $C1F1
TEST1         $7D
START         $81
BEG           $85
TEMP2         $89
TEMP1         $8D
S10           $C102
MO1           $C114
BYTE1         $C129
SKIP          $C152
DONE          $C15F
B10           $C196
TEST          $C1CE
IOPTR         $C1D9
INPTR         $C1E0
DOPTR         $C1E7
```

Den Object-Code können Sie mit dem Supermon 64 oder einem beliebigen anderen Monitor eingeben.

Zufallszahlen-Generator

Der Zufall ist ein wichtiger Bestandteil von vielen Spielen, z.B. Karten-, Aktion- oder Abenteuerspielen.

Das Programm beruht auf der Benutzung eines Pseudo-Zufall-Shift-Registers. (RNDM und RNDM+1). Mindestens eines der beiden Register muß den Wert Null besitzen. Wir haben die beiden Speicheradressen $FB und $FE aus Seite 0 gewählt. Bevor Sie das Programm starten, müssen Sie mit Hilfe des Monitors eines dieser Register auf Null setzen.

Nach dem Assemblieren können Sie das Programm über den Monitor durch GOTO C100 starten.

Das folgende Programm erzeugt nur eine Zufallszahl. Wenn es von BASIC aufgerufen werden soll, muß BRK am Ende durch RTS ersetzt werden.

```
                *RANDOM

                          ORG $C100
                CHROUT    EQU $FFD2
                RNDM      EPZ $FB
C100: A5FE      RANDOM    LDA $FE           ;SET ITERATIONS
C102: 48        R1        PHA               ;SAVE COUNTER
C103: A5FB                LDA RNDM          ;GET BYTE
C105: 2A                  ROL
C106: 45FB                EOR RNDM          ;XOR BITS 13 & 14
C108: 2A                  ROL
C109: 2A                  ROL
```

```
C10A: 26FC            ROL RNDM+1       ;SHIFT BYTE
C10C: 26FB            ROL RNDM         ;SHIFT 2. BYTE
C10E: 68              PLA              ;GET COUNTER
C10F: 18              CLC
C110: 69FF            ADC #$FF         ;DECREMENT
C112: D0EE            BNE R1           ;IF NOT DONE DO AGAIN
C114: A5FB            LDA RNDM         ;GET RANDOM BYTE
C116: 20D2FF          JSR CHROUT       ;PRINT
C119: 00              BRK

PHYSICAL ENDADDRESS: $C11A

*** NO WARNINGS

CHROUT          $FFD2             RNDM          $FB
RANDOM          $C100   UNUSED    R1            $C102
```

Das folgende Programm erzeugt ebenfalls Zufallszahlen, aber es gibt immer 10 Zufallszahlen aus.

Achtung! Wenn Sie weniger als 10 Zeichen auf dem Bildschirm sehen, dann sind Steuerzeichen darunter (z. B. Carriage Return), die direkt ausgeführt werden.

```
            *RANDOM10

                      ORG $C100
            CHROUT    EQU $FFD2
            RNDM      EPZ $FB
            COUNTER   EPZ $FD
C100: A900            LDA #0
C102: 85FD            STA COUNTER
C104: A5FE  RANDOM    LDA $FE          ;SET ITERATIONS
C106: 48    R1        PHA              ;SAVE COUNTER
C107: A5FB            LDA RNDM         ;GET BYTE
C109: 2A              ROL
C10A: 45FB            EOR RNDM         ;XOR BITS 13 & 14
C10C: 2A              ROL
C10D: 2A              ROL
C10E: 26FC            ROL RNDM+1       ;SHIFT BYTE
C110: 26FB            ROL RNDM         ;SHIFT 2. BYTE
C112: 68              PLA              ;GET COUNTER
C113: 18              CLC
C114: 69FF            ADC #$FF         ;DECREMENT
C116: D0EE            BNE R1           ;IF NOT DONE DO AGAIN
```

```
C118: A5FB          LDA RNDM        ;GET RANDOM BYTE
C11A: 20D2FF        JSR CHROUT      ;PRINT
C11D: E6FD          INC COUNTER
C11F: A90B          LDA #$0B
C121: C5FD          CMP COUNTER
C123: D0DF          BNE RANDOM
C125: 00            BRK

PHYSICAL ENDADDRESS: $C126

*** NO WARNINGS

CHROUT       $FFD2          RNDM       $FB
COUNTER      $FD            RANDOM     $C104
R1           $C106
```

Notizen

Zugriff auf Maschinenprogrammme von BASIC aus

Der BASIC-Programmierer will oft ein Programm schneller machen. Die beste Möglichkeit bieten dabei Maschinenprogramme, die in das BASIC-Programm eingebunden werden.

In diesem Fall muß das Maschinenprogramm an einen sicheren Platz (sicher vor BASIC) abgelegt werden.

Von BASIC aus kann ein Maschinenprogramm mit folgenden Befehlen aufgerufen werden:

5 SYS(X) oder 10 A=USR(X)

X ist die dezimale Anfangsadresse des Maschinenprogramms.

Wenn Sie z. B. ein Maschinenprogramm an der Adresse $C100 von BASIC aus aufrufen wollen, muss der Befehl SYS(49152) heissen. Verwenden Sie jedoch den Befehl USR(X), so muss die Adresse vorher in den Speicherzellen 785 (Lower Byte)und 786 (Higher Byte) festgelegt werden.
Beispiel:
Sie haben zwei Maschinenprogramme im Speicher. Eins an

der Stelle $C000 und das andere an der Stelle $C800. Das erste soll aufgerufen werden,wenn die Variable V V kleiner 10 ist , die andere Routine soll aufgerufen werden, wenn V größer oder gleich 10 ist. Das nachfolgende BASIC Programm könnte diese Aufgabe lösen, vorausgesetzt, die Maschinenprogramme befinden sich auch im Speicher.

```
200 IF V>9 THEN 230
210 POKE 785,0:POKE786,192
220 X=USR(0):GOTO 250
230 POKE 785,0:POKE 786,200
240 X=USR(0)
:
:
```

Der Befehl USR(x) erlaubt die Übergabe eines Parameters in ein Maschinenprogramm. Der Befehl X=USR(10) übergibt z. B. die Zahl 10 über den Fliesskommaakkumulator an das Maschinenprogramm. Der Floating Point Akku beginnt beim C-64 an der Adresse $61. Umgekehrt kann auch vom Maschinenprogramm eine Zahl an ein BASIC Programm übergeben werden. Das Maschinenprogramm muss die Zahl nur in den Akku legen,bevor es nach BASIC zurückkehrt. Achtung! In beiden Fällen muss das Maschinenprogramm mit RTS abgeschlossen sein.

Wo soll man Maschinenprogramme ablegen?
Wie schon vorher erwähnt,eignet sich beim C-64 besonders der Bereich ab Adresse Hex $C000 bis Adresse $CFFF (49152 bis 53247). Wenn dies nicht ausreicht, mussen Sie noch Speicherplatz vom normalen BASIC Bereich stehlen. Dies geschieht dadurch, dass Sie einen neuen oberen Bereich für Basic festlegen. Er ist normalerweise bei $9FFF. Um diese Adresse zu verändern,müssen Sie die neue Adresse in die Zellen 51,52 und 55 und 56 dez. in der Zeropage poken.
Beispiel:
Angenommen Sie brauchen noch 2k RAM so müssen Sie folgendes im Direktmodus eingeben:
POKE 51,0:POKE 52,152:POKE 55,0:POKE 56,152:CLR
Dies erlaubt Ihnen dann bis zu 6k RAM für Maschinenprogramme zu verwenden.
$9800-$9FFF und $C000-$CFFF

Die KERNAL-Routinen

Die meisten Programme in diesem Buche enthalten Unterprogrammspruenge in bestimmte, im C-64 bereits residente Programmteile. Die beiden Routinen CHRIN und CHROUT treffen wir bei diesen Programmen am meisten an. Diese beiden Unterprogramme befinden sich im Kernal, das ist ein ROM Bereich am oberen Ende des Speichers. Nachfolgend wollen wir Ihnen noch eine kurze Beschreibung der wichtigsten Unterprogramme geben.

CHRIN , Eingabe eines Zeichens ($FFCF)
Dieses Programm wartet auf eine Eingabe von einem Eingabegeraet her. (Eingabegeraet=Device) Wenn nichts anderes vorher festgelegt wurde, ist mit dem Eingabegeraet die Tastatur gemeint.Alle Zeichen, die in diesem Falle eingegeben werden, werden in den System Eingabepuffer ab $0200 abgelegt. Wenn die Unterroutine verlassen und zum Hauptprogramm zurueckgekehrt wird, wird das zuletzt durch ein Carriage RETURN abgeschlossene Zeichen in den Akkumulator uebernommen.

CHROUT, Ausgabe eines Zeichens ($FFD2)
Diese Routine schickt den Inhalt des Akkumulators (als ASCII Zeichen) an ein Device. Device=Ausgabegeraet). Wenn nichts anderes vorher festgelegt wurde, ist dieses Ausgabegeraet der Bildschirm. Wenn Sie z.B. ein "A" auf dem Bildschirm ausgeben wollen, muessen Sie folgendes kleine Programm schreiben:

```
LDA #$41
JSR CHROUT
```

Wichtig ist, dass Sie wissen, dass die Ausgabe des Zeichens mit dieser Routine an der momentanen Cursorposition erfolgt.

Quelltext fuer ein Beispiel zu CHROUT

Gibt 255 Hex Bytes aus

```
                      ORG $C100
               PRTBYT EQU $C000
C100: A200            LDX #$00
C102: BD00C0 MARKE    LDA $C000,X
C105: 2000C0          JSR PRTBYT
C108: E8              INX
C109: D0F7            BNE MARKE
C10B: 00              BRK

PHYSICAL ENDADDRESS: $C10C

*** NO WARNINGS

PRTBYT            $C000
MARKE             $C102


             BYTE     EQU     $C023
             CHROUT   EQU     $FFD2
                      ORG     $C000
C000: 8D23C0 PRTBYT   STA     BYTE
C003: 4A              LSR
C004: 4A              LSR
C005: 4A              LSR
C006: 4A              LSR
C007: 2014C0          JSR     OUTPUT
C00A: AD23C0          LDA     BYTE
C00D: 2014C0          JSR     OUTPUT
C010: AD23C0          LDA     BYTE
C013: 60              RTS
C014: 290F   OUTPUT   AND     #$0F
C016: C90A            CMP     #$0A
C018: 18              CLC
C019: 3002            BMI     $C01D
C01B: 6907            ADC     #$07
C01D: 6930            ADC     #$30
C01F: 4CD2FF          JMP     CHROUT
C022: 60              RTS


PHYSICAL ENDADDRESS: $C023
```

```
*** NO WARNINGS

BYTE            $C023
PRTBYT          $C000    UNUSED
CHROUT          $FFD2
OUTPUT          $C014
```

GETIN, Eingabe eines Zeichens ($FFE4)
Diese Routine holt ein Zeichen aus dem Tastaturpuffer. Dieser kann beim C-64 bis zu 10 Zeichen enthalten. Wenn der Puffer leer ist, erscheint eine Null.

Eingabe und Ausdruck von Zeichen

```
                         ORG $C100
               GETIN     EQU $FFE4
               CHROUT    EQU $FFD2
C100: 20E4FF   INIT      JSR GETIN
C103: C903               CMP #$03
C105: F006               BEQ ENDE
C107: 20D2FF             JSR CHROUT
C10A: 4C00C1             JMP INIT
C10D: 00       ENDE      BRK

PHYSICAL ENDADDRESS: $C10E

*** NO WARNINGS

GETIN           $FFE4
INIT            $C100
CHROUT          $FFD2
ENDE            $C10D
```

Druckt ASCII-Zeichen auf Bildschirm

```
                         ORG $C100
               CHROUT    EQU $FFD2
C100: A220     INIT      LDX #$20
C102: 8A       LOOP      TXA
C103: 20D2FF             JSR CHROUT
```

```
C106: E8          INX
C107: E060        CPX #$60
C109: D0F7        BNE LOOP
C10B: 00          BRK
C10C: 4C00C1      JMP INIT

PHYSICAL ENDADDRESS: $C10F

*** NO WARNINGS

CHROUT        $FFD2
LOOP          $C102
INIT          $C100
```

PLOT , Platzieren des Cursors an einer bestimmten Stelle auf dem Bildschirm.
Diese Kernal Routine erlaubt es Ihnen, den Cursor auf dem Bildschirm an eine ganz bestimmte Stelle zu bringen. Mit Hilfe dieser Routine laesst sich aber auch die momentane Cursor Position auslesen. Wenn Sie diese Routine aufrufen und das Carry Flag vorher geloescht haben, dann liefern das X-Register und das Y-Register die Koordinaten der Cursorposition.
Y-Register liefert die Reihe

X-Register liefert die Spalte
Wenn Sie jedoch vorher die gewuenschten Werte fuer Reihe und Spalte in die Register bringen und das Carry Bit loeschen und dann anschliessend in die PLOT Routine springen, erscheint der Cursor an der gewuenschten Stelle auf dem Bildschirm.

Beispiel:

```
LDY #$05
LDX #$08
CLC
JSR PLOT
```

```
                    PRTBYT      EQU $C000
                                ORG $C100
                    PLOT        EQU $FFF0
                    CHROUT      EQU $FFD2
C100: A005          MARKE       LDY #$05
C102: A208                      LDX #$08
C104: 18                        CLC
C105: 20F0FF                    JSR PLOT
C108: A941                      LDA #$41
C10A: 20D2FF                    JSR CHROUT
C10D: 00                        BRK




PHYSICAL ENDADDRESS: $C10E

*** NO WARNINGS

PRTBYT              $C000     UNUSED
CHROUT              $FFD2
PLOT                $FFF0
MARKE               $C100     UNUSED
```

RDTIM, Lese die interne Systemuhr,($FFDE)
Dieses Unterprogramm liefert die momentane Zeit der Systemuhr Ihres C-64. Drei Byte werden geliefert, wobei das hoeherwertige Byte im Akkumulator steht, das naechste im X-Register und das niederwertige Byte im Y-Register steht.

SETTIM, Stellen der Zeit fuer die Systemuhr.($FFDB)
Diese Routine stellt die interne Uhr Ihres C-64. Der Wert wird wie in RDTIM ueber den Akkumulator und das X- und Y-Register festgelegt.

Notizen

Kapitel A

Zahlensysteme,

In diesem Kapitel wollen wir, basierend auf tägliche Erfahrungen, einige mathematische Grundlagen herausarbeiten, die es wesentlich einfacher machen, das Innenleben eines Computers besser zu verstehen.

Dezimalzahlen

Zahlengröße

Binärzahlen, Bits und Bytes

Hexadezimalzahlen

Dezimalzahlen und das Größenkonzept

Die westliche Welt hat zur Darstellung der verschiedenen Größen die zehn arabischen Symbole: 0,1,2, 3,4,5,6,7,8 und 9 übernommen. Diese Werte können auch noch anders wiedergegeben werden:

z. B. "drei" kann als three (englisch), 3 trois (französisch), III (römisch), etc. dargestellt werden.

Mit Ausnahme der römischen Zahlen sind die obigen Beispiele Dezimalzahlen; sie gehören also zu dem 10er-System, mit dem wir täglich arbeiten. Das 10er-Stystem, zeichnet sich durch die 10 Ziffern aus, mit denen die

Zahlenmengen geschrieben werden können. Für große (mehrstellige) Zahlen kombinieren wir einige Ziffern; z. B. geben wir den Inhalt einer Eierschachtel mit "12" an. Die rechte Ziffer wird nun Einer, die nächste Zehnerstelle genannt. Die Zehnerstelle gibt an, wieviel Zehnergruppen Eier vorhanden sind. Die Gesamtmenge wird wie folgt berechnet: 10 multipliziert mit der Ziffer in der Zehnerstelle plus Anzahl in der Einerstelle.

Wäre noch eine weitere Ziffer auf der linken Seite vorhanden, so müßte man diese Ziffer zweimal mit 10 multiplizieren (d.h. mit 100).

Das 10er- oder Dezimalsystem ist also durch folgende Faktoren gekennzeichnet:

1.) Es gibt 10 Ziffern (0-9)

2.) Jede Stelle links von der Einerstelle muß mit einem Multiplikator, der jeweils um den Faktor 10 wächst multipliziert werden.

3.) Dezimalzahlen sind nicht die einzige Möglichkeit Mengen darzustellen.

Wir wollen uns jetzt mit anderen Zahlensystemen, die bei Computern eingesetzt werden, beschäftigen. (Für uns sind sie schwieriger, aber für den Computer einfacher!!!)

Binärzahlen...

Im allgemeinen verarbeiten die Computer nicht direkt die Symbole des Dezimalsystems. Der Computer besteht aus Schaltkreisen, die nur zwei logische Zustände einnehmen

können (im Gegensatz zu den zehn des Dezimalsystems). Die Schaltkreise im Computer kennen nur ein Symbol mit hoher Spannung (ca. 5V) und ein anderes mit niedriger Spannung (ca. 0V). Diese Zustände werden oft mit "High" oder "1" für den hohen und mit "Low" oder "0" für den niedrigen Spannungswert bezeichnet. Mehrstellige Binärzahlen können deshalb durch mehrere Kreise dargestellt werden, die entweder den Zustand "0" oder "1" einnehmen. Um eine Parallele zum 10er-System zu ziehen, können wir diese nun als Binärsystem definieren.

Dazu die kennzeichnenden Eigenschaften:

1.) Das Binärsystem besteht aus 2 Symbolen (0, 1)

2.) Jede Stelle links neben der Einerstelle muß mit einem Multiplikator, der jeweils um den Faktor 2 wächst, multipliziert werden.

Stellenwerte der Position bei Dezimalzahlen und bei Binärzahlen:

Dezimal (10000er) (1000er) (100er) (10er) (1er)

Binär (16er) (8er) (4er) (2er) (1er)

Wir bringen jetzt einige Beispiele für Binärzahlen:

WERT	BINÄR	ERKLÄRUNG DER BINÄRZAHL
Null	0	0 auf Einerstelle
Eins	1	1 auf Einerstelle
Zwei	10	2 mal 1 auf Zweierstelle und 0 auf Einerstelle
Drei	11	2 mal 1 auf Zweierstelle und 1 auf Einerstelle
Vier	100	2 mal 2 mal 1 auf Viererstelle und 2 mal 0 auf Zweierstelle und 0 auf Einerstelle
Fünf	101	wie oben, nur 1 auf Einerstelle
Dreizehn	1101	wie oben, nur und 2 mal 2 mal 2 mal 1 auf Achterstelle

Es gibt keinen Trick, der das Lesen von Binärzahlen direkt ermöglicht. Wenn Sie den dezimalen Wert einer Binärzahl wissen wollen führt kein Weg daran vorbei, die jeweiligen Bits mit 1,2,4,8,16 etc. zu multiplizieren und zum letzten Ergebnis dazu zu addieren

BINARY NUMBER SYMBOL	DECIMAL NUMBER SYMBOL
110	6
101000	40
1000000	64
111111	63
111110	62
111101	61
1111111	127

Eine Stelle einer Binärzahl oder eines Schaltkreises im Computer kann immer nur einen Zustand von zwei Möglichkeiten anzeigen. Das bedeutet, daß eine einzelne Stelle nicht viel Information bringt. Wir bezeichnen deshalb eine Stelle einer Binärzahl als Bit. Ein Bit kann entweder 0 oder 1 sein. Ein Byte besteht aus acht solchen Bits.

Man muß sich nun im Klaren darüber sein, daß Binärzahlen einfach eine andere Darstellungsmöglichkeit von Zahlen sind, genauso wie z.B. römische Zahlen. In allen Fällen gibt ein vorhandenes Symbol nur einen bestimmten Wert wieder; wie wir diesen Wert schreiben ist dann zweitrangig.

Hexadezimalzahlen

Die vorhergegangenen Überlegungen zeigen, daß Binärzahlen bei großen Zahlenwerten sehr unübersehbar werden. Das ergibt sich aber zwangsläufig daraus, daß nur zwei Symbole zur Verfügung stehen. Im Dezimalsystem hatten wir 10 Symbole zur Darstellung und große Zahlenwerte konnten mit wenig Stellen wiedergegeben werden. Ideal wäre nun ein Zahlensystem, was uns eine große Anzahl von Symbolen bietet, aber eine möglichst einfache Verbindung zum Binärsystem des Computers darstellt.

Beachten Sie, daß eine Vierbitzahl einen Wert von 0 (0000) bis 15 (1111) annehmen kann, d. h. 16 mögliche Kombinationen. Stellen sie sich jetzt vor, daß wir für jede dieser Kombinationen einen Buchstaben oder eine Zahl einsetzen (siehe rechte Spalte der nächsten Tabelle).

DECIMAL NUMBER	BINARY NUMBER	HEXADECIMAL NUMBER
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Wundern Sie sich nicht, daß Buchstaben zur Darstellung von Zahlen herangezogen werden, aber es handelt sich hierbei um eine reine Definitionssache. Wenn jetzt definiert wird, daß das Symbol "D" den Wert 13 haben soll, dann ist das eben so. Die 16 Symbole oder (0-9, A-F) sind die 16 Symbole des Hexadezimalsystems. Für mehrstellige Hexadezimalzahlen fangen wir wieder bei der Einerstelle an. Aber jedesmal, wenn wir eine Stelle nach links gehen, addieren wir eine Multiplikation mit 16 dazu.

DEZIMAL	BINÄR	HEXA-DEZIMAL	ERKLÄRUNG
15	1111	F	15 auf Einerstelle
16	1 0000	10	1 auf 16er-Stelle
17	1 0000	11	1 auf 16er-Stelle + 1 auf Einerstelle
42	10 1010	2A	2 auf 16er-Stelle + 10 auf Einerstelle
255	1111 1111	FF	15 auf 16er-Stelle + 15 auf Einerstelle
256	1 0000 0000	100	1 auf 156er-Stelle + 0 auf Einer- und 16er-Stelle
769	11 0000 0001	301	3 auf 256er-Stelle + 0 auf 16er-Stelle + 1 auf Einerstelle
783	11 0000 1111	30F	3 auf 256er-Stelle + 0 auf 16er-Stelle + 15 auf Einerstelle

Kennzeichen des Hexadezimalsystems:

1.) 16 Symbole (0-9, A-F)

2.) Jede Stelle links von der Einerstelle muß mit einem Multiplikator, der jeweils um den Faktor 16 wächst

multipliziert werden. (d.h. Multiplikatoren sind: 1, 16, 256, 4096 etc.)

Beachten Sie, daß die binäre Darstellung leicht in die hexadezimale Form umgewandelt werden kann:

1.) Teilen Sie die Binärzahl in Vierbitzahlen auf.

2.) Schreiben Sie für jede Vierbitzahl das hexadezimale Symbol an.

3.) Sie können hexadezimale Zahlen in Binärzahlen umrechnen, in dem Sie den oberen Vorgang einfach umdrehen.

Die hexadezimale Darstellung bringt also eine Aussage über die binäre Darstellung im Computer.

Bei mehrstelligen Zahlen ist es nicht immer eindeutig, um welche Darstellungsart es sich handelt. "1101" kann als Binärzahl (13), als Dezimalzahl (1101) oder als Hexadezimalzahl (4315) gelesen werden. Dagegen ist "1301" eindeutig keine Binärzahl. (Aber es könnte eine Dezimal- oder Hexadezimalzahl sein)

In solchen Fällen wird im Allgemeinen angegeben, um welche Darstellung es sich handelt. Entweder durch den Index "2" oder das Wort "Binär". Hexadezimalzahlen werden oft durch ein vorgestelltes "$" oder ein nachgestelltes "H" gekennzeichnet. (z.B. $FFFF, 4020H)

"$" wird bei den meisten 6510/6502-Systemen verwendet.

Übungen zu Kapitel A

1.) Rechnen Sie folgende Binärzahlen in Dezimalzahlen um:

```
1111 1111
0111 1111
 111 1111
   1 0000
1000 1000
0100 0101
1111 1110
```

Antworten: 255, 127, 127, 18, 136, 69, 254)

2.) Rechnen Sie die Binärzahlen aus Bsp. 1.) in Hexadezimalzahlen um.

(Antworten: $FF, $7F, $7F, $10, $88, $45, $FE)

Das folgende Unterprogramm in Maschinensprache rechnet Hexadezimalzahlen in Dezimalzahlen um.

1. Listing

```
              OUT LNM,3
              ORG $C100
              STA $02
              STX $03
              LDA #$00
              STA $04
              STA $05
              STA $06
              SED
              LDY #$10
    LOOP2     LDX #$03
              ASL $03
              ROL $02
    LOOP1     LDA $03,X
              ADC $03,X
              STA $03,X
              DEX
```

```
        BNE LOOP1
        DEY
        BNE LOOP2
        CLD
        LDA $04
        LDX $05
        LDY $06
        RTS
```

2. Listing

```
                                ORG $C100
C100: 8502                      STA $02
C102: 8603                      STX $03
C104: A900                      LDA #$00
C106: 8504                      STA $04
C108: 8505                      STA $05
C10A: 8506                      STA $06
C10C: F8                        SED
C10D: A010                      LDY #$10
C10F: A203      LOOP2           LDX #$03
C111: 0603                      ASL $03
C113: 2602                      ROL $02
C115: B503      LOOP1           LDA $03,X
C117: 7503                      ADC $03,X
C119: 9503                      STA $03,X
C11B: CA                        DEX
C11C: D0F7                      BNE LOOP1
C11E: 88                        DEY
C11F: D0EE                      BNE LOOP2
C121: D8                        CLD
C122: A504                      LDA $04
C124: A605                      LDX $05
C126: A406                      LDY $06
C128: 60                        RTS


PHYSICAL ENDADDRESS: $C129

*** NO WARNINGS

LOOP2               $C10F
LOOP1               $C115
```

Das erste Quelltext Listing zeigt Ihnen den Ausdruck des Editors. So muss das Listing auch in Ihren Editor/Assembler eingegeben werden.OUT LNM,3 assembliert im MACROFIRE auf den Bildschirm.Das zweite Listing ist eine Ausgabe des Assemblers über den Drucker. Ein sog. Assemblerlisting.Hieraus entnehmen Sie auch den HEXCODE, wenn Sie das Programm z.B. mit einem Monitor eintippen wollen.Es steht, wie alle unsere Beispiele ab Adresse $C100.

Anwendung des Beispiels:

Die Hexadezimalzahl muss im Akkumulator stehen (Hoeherwertiges Byte) und das niederwertige Byte muß im X-Register stehen. Dann kann in das HEX-DEZ Programm gesprungen werden.

Beispiel:

Geben Sie das Programm in Ihren C-64. Entweder mit einem Monitor oder mit einem symbolischen Assembler. Das Programm liegt ab Adresse $C100. Wenn Sie mit MACROFIRE arbeiten, geben Sie den Quelltext ein und assemblieren Sie an die Adresse $C100.Gehen Sie dann mit dem Befehl <CTRL>-<P> in den eingebauten Monitor und sehen sich das Programm (Hexcode) ab Adresse $C100 an.

Um das Programm auch wirklich testen zu können, müssen wir noch ein kleines Hilfsprogramm eingeben. Geben Sie also das folgende Programm ab $C000 mit dem Monitor in Hex ein:

```
C000 A9
C001 10
C002 A2
C003 1F
C004 20
C005 00
C006 C1


C007 00
```

Starten Sie dieses kleine Programm mit G C000. Es lädt nun die Hexadezimalzahl $101F in den Akku und das X-Register und springt in die HEX-DEZ Routine. Danach läuft das Programm auf den BRK Befehl. Dieser bewirkt im Monitor einen Programmstop mit Ausgabe der Register auf dem Bildschirm. Dort finden wir nun im X- und Y-Register die errechnete Dezimalzahl 4127 dez. 101F hex=4127 dez

Die Hexadezimalzahl muß sich im Akkumulator (höherwertiges Byte) und im X-Register (niedriger-wertiges Byte) befinden, wenn das Unterprogramm aufgerufen wird.

z.B.: Umrechnung von 101F in dezimalen Wert:

```
A9 10     LDA #$10
A21F      LDX #$1F
20 00 06  JSR $0600
00        BRK
```

Wenn der Atmona einen BRK antrifft, wird der Inhalt der Register angezeigt. Der dezimale Wert ist im X- und Y-Register.

101F hex = 4127 dez

Notizen

Kapitel B

Kapitel B: Das Digitalkonzept

In diesem Kapitel geben wir einen Überblick über das logische Digitalkonzept und über die Arten elektronischer Bauteile, die verwendet werden, um logische Operationen auszuführen und Daten im Computer zu speichern.

Logik bei der Programmierung und bei der Hardware

Logische Operationen und logische Gatter

Logische Schaltungen und Decoder

Decoder und Speicher

NICHT-UND-, WEDER-NOCH-, UND-, EXCLUSIV-ODER-Gatter

Logik bei der Programmierung und bei der Hardware.

Man programmiert einen Computer, um eine Reihenfolge logischer Operationen auszuführen. Ein Programm besteht aus einer Reihenfolge von Anweisungen für den Computer. Oft wollen wir den Computer so programmieren, daß er Entscheidungen treffen kann, die von Informationen abhängen, die er von der Außenwelt erhält. z.B. soll ein Computer die Schranken an einem Bahnübergang steuern. Sensoren müßen feststellen, ob sich ein Zug nähert. Das Programm hat dann folgende Form.

1. Programmanfang

2. Kommt ein Zug?

3. Wenn ja, gehe zu 5.

4. Gehe zu 2.

5. Schranke unten?

6. Wenn nein dann Schranke nach unten

7. Zug noch da oder kommt anderer Zug?

8. Wenn ja, gehe zu 7.

9. Schranke nach oben

10. Gehe zu 2.

Dieses Programm öffnet oder schließt die Schranke entsprechend der Information, die es von den Sensoren bekommt. Ein anderes Beispiel ist der Wortprozessor, mit dem dieser Text geschrieben worden ist. Das Programm muß immer entscheiden, welche Taste gedrückt wurde, d.h. welcher Buchstabe eingegeben wurde.

Bei den beiden Beispielen müssen aber auch Daten an die Außenwelt zurückgegeben werden. Das Schrankenprogramm muß die Schranken heben oder senken, der Wortprozessor muß den Text auf den Bildschirm ausgeben, wenn er eingetippt wird. Er kann den Text aber auch speichern, wieder laden (z. B. zum Verbessern) und ausdrucken, je nach dem, wie es der Benutzer will.

In jedem Fall aber führt der Computer eine Reihenfolge logischer Operationen aus. In diesem Kapitel wollen wir uns nun mit den Vorgängen im Inneren des Computers, mit der Hardware, beschäftigen.

Logische Operationen und logische Gatter.

Beachten Sie folgende Anweisungen:

Wenn (A=wahr), dann (Z=wahr)

Wenn (A=falsch), dann (Z=falsch)

A, Z sind immer wahr oder falsch; Zwischenzustände sind nicht möglich. Durch die beiden Anweisungn haben wir Z eindeutig definiert und zwar für alle Möglichkeiten von A. Wir wollen dies jetzt mit Hilfe eines elektronischen Schaltkreises nachvollziehen.

Wir definieren:

1. Wahr wird durch eine Spannung von +2V - +5V (d.h. HIGH) symbolisiert.

2. Falsch wird durch eine Spannung von 0V = +0, 5V (d. H LOW) symbolisiert.

Stellen Sie sich jetzt ein Stück Draht vor. Das eine Ende wird "Eingang-A", das andere "Ausgang-Z" genannt. Wir haben damit ein Modell unserer logischen Anweisung:

1. Wenn (A=HIGH) dann (Z=HIGH).
Eine bei A angelegte Spannung "HIGH" ergibt die gleiche Spannung "HIGH" bei Z.

2. Wenn (A=LOW), dann (Z=LOW)
Auch hier wird eine Spannung LOW uebertragen. Betrachten Sie dagegen folgende Anweisungen:

1. Wenn (A=wahr), dann (Z=falsch)

2. Wenn (A=falsch), dann (Z=wahr)

Diese etwas komplizierte Situation koennen wir nicht mehr durch ein Stückchen Draht nachvollziehen.

Wir müssen hier eine "NICHT-Schaltung" oder Negationsschaltung verwenden. Diese Schaltung wird von vielen Firmen angeboten, aber wir brauchen uns jetzt nur eine "Black Box" vorzustellen, die zwei Anschlüsse hat (Eingang A, AusgangZ). Eine hohe/niedrige Spannung bei A ergibt eine niedrige/hohe Spannung bei Z. Bei Z darf kein Eingangssignal anliegen; es ist aber möglich, an Z eine zweite "Nicht-Schaltung" anzuschließen. Das Siganl nach dem zweiten Ausgang wäre identisch mit dem ersten Eingangssignal.

Es gibt ein Standardsymbol, um eine Negationsschaltung zu zeichnen. (Abbildung 2.1)

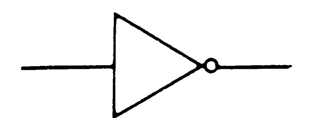


Abbildung 2.1.
Negationsschaltung

Es gibt auch ein Symbol, welches für eine Schaltung steht, die sich wie unser Stück Draht verhält. (Abbiludng 2.2.) Beachten Sie, daß hier der Kreis beim Ausgang fehlt. Dieser symbolisiert nämlich die Negation.

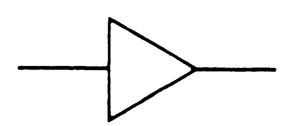


Abbildung 2.2
Logischer Buffer

In verschiedenen Situationen sollen vielleicht die Eingänge von verschiedenen logischen Gattern zu einem Ausgang eines logischen Gatters zusammengeschlossen werden. Um zu verhindern, daß der Ausgang zu hoch wird, werden logische Buffer verwendet.

Eine andere Anweisung:

Wenn (A=wahr), oder (B=Wahr), dann (Z=wahr).
(Anderenfalls Z=falsch)

Hier handelt es sich um einen Ausgang (Z) bei zwei Eingängen (A, B). Üblicherweise werden die möglichen Kombinationen in einer Wahrheitstabelle angegeben:

INPUT A	INPUT B	OUTPUT Z
0	0	0
0	1	1
1	0	1
1	1	1

Abbildung 2.3
Wahrheitstabelle: Z= (A ODER B)

Bei der beschriebenen Funktion handelt es sich um ein ODER-Gatter mit zwei Eingängen.

Dafür nun das Symbol:

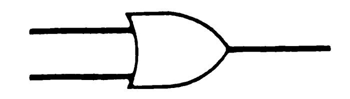


Abbildung 2.4 A
ODER-Gatter mit zwei Eingängen

Es sind auch ODER-Gatter mit 3 oder mehr Eingängen möglich:

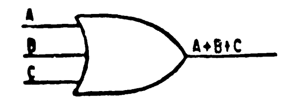
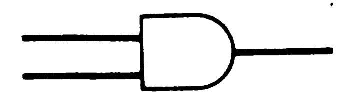


Abbildung 2.4 B
ODER-Gatter mit drei Eingängen

Ein Computer besteht aus sehr vielen dieser ODER-Gatter.



Man kann aber noch andere logische Operationen definieren:

Wenn (A=wahr), und (B=wahr), denn (Z=wahr)

Wie bei den ODER-Gattern können auch bei diesen UND-Gattern mehr als zwei Eingänge vorhanden sein.

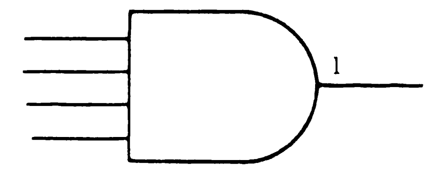


Abbildung 2.5.B
Und-Gatter mit 4 Eingängen

Wir haben also im Prinzip drei logische Gatter (UND,ODER, NICHT). Diese Gatter sind üblicherweise in Kunststoffgehäusen mit den entsprechenden Anschlüssen.

Zusätzlich müssen mindestens zwei Anschlüsse zur Stromversorgung der internen Schaltkreise vorhanden sein. In der weitverbreiteten "Transistor-Transistor-Logik"-Familie (TTL), die wir hier behandeln, werden Spannungen über 2V als "wahr" oder "1" und Spannungen unter 0, 5V als "falsch" oder "0" angesehen. Spannungen zwischen 0, 5V und 2V sind nicht erlaubt und können nicht gelesen werden. Spannungen, die höher als 5V oder negativ sind, führen zur Zerstörung des Bausteins. Die Ausgänge führen ebenso nur diese Spannungen.

Logische Schaltungen und Decoder

Problem: Vier Signale sind gegeben (A,B,C und D), die auf vier verschiedenen Leitungen zugeführt werden. Es soll eine Schaltung entworfen werden, die logisch "1" ausgibt, wenn gilt: ABCD = 1010 (d.H.A=1, B=0 ...)

Lösung: Wir bezeichnen unseren Ausgang mit "Z"
Wir wollen folgende Schaltungen bauen:

Wenn (A=wahr) und (B=falsch) und (C=wahr) und (D=falsch), dann (Z=wahr)

Wegen B und D ist es unmöglich, dies mit einem UND-Gatter mit vier Eingängen zu erreichen. Wenn wir aber B und D invertieren (NICHT) können wir zwei neue Signale definieren:
M = NICHT B
N = NICHT D

daraus ergibt sich:

Wenn (A=war) und (M=wahr) und (C=wahr) und (N=wahr, dann (Z=wahr)

Nun die Schaltung:

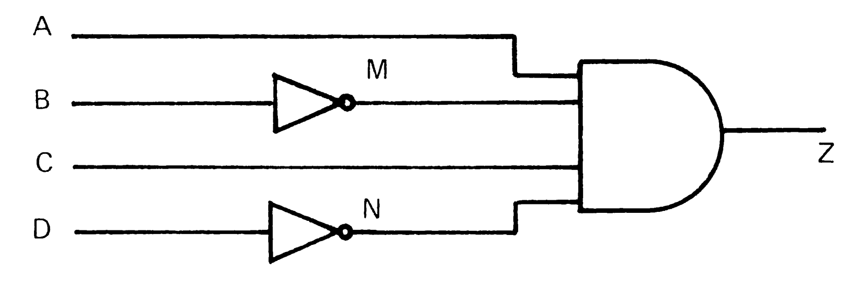


Abbildung 2.6
Beispielschaltung

In Abbildung 2. 6 sehen Sie ein Beispiel für eine Decoderschaltung. Die Schaltung decodiert eine komplexe Eingabe und erzeut ein kennzeichnendes Signal für eine bestimmte Eingabe. Wenn die Eingabe ABCD einer Vierbitbinärzahl entspricht, dann decodiert unsere Decoderschaltung den Dezimalwert "10".

Erinnern Sie sich, daß einstellige Binärzahlen 16 Kombinationen ermöglichen (0-15). Es ist möglich einen Decoder zu bauen, der vier Eingängen und 16 Ausgängen hat. Jedes Ausgangssignal würde dann genau eine der Möglichkeiten darstellen. Wenn die Eingänge denn eine und nur eine der 16 Möglichkeiten darstellen, so führt genau ein Ausgang das Signal "wahr". Die Wahrheitstabelle für so eine Schaltung zeigt Abbildung 2. 7. Die Schaltung zeigt Abbildung 2.8. Die Eingänge sind A0-A3; die Ausgänge Y0-Y5.

Zusammenfassung der Verknüpfungsschaltungen

Zusammenfassung

Da vor allem am Anfang noch manchmal Zweifel über die Logik der Grundbausteine herrschen, ist im Folgenden der wichtigste Sachverhalt des bisher behandelten Stoffes zusammengetragen. Auch zum schnellen Nachschlagen wird sich die Tabelle als nützlich erweisen.

	Inverter	AND	NAND	OR	NOR	EX-OR	EX-NOR
alte Norm	E, A	E_1, E_2, A	E_1, E_2, A	E_1, E_2, A	E_1, E_2, A	E_1, E_2, e, A	E_1, E_2, e, A
neue Norm	1	&	&	≥ 1	≥ 1	= 1	=
amerikan. Norm							
Boole'sche Funktionsgleichung	$A = \overline{E}$	$A = E_1 \wedge E_2$	$A = \overline{E_1 \wedge E_2}$	$A = E_1 \vee E_2$	$A = \overline{E_1 \vee E_2}$	$A = E_1 / E_2$	$A = \overline{E_1 / E_2}$
Logik-Tafel	E A 1 0 0 1	E_1 E_2 A 0 0 0 0 1 0 1 0 0 1 1 1	E_1 E_2 A 0 0 1 0 1 1 1 0 1 1 1 0	E_1 E_2 A 0 0 0 0 1 1 1 0 1 1 1 1	E_1 E_2 A 0 0 1 0 1 0 1 0 0 1 1 0	E_1 E_2 A 0 0 0 0 1 1 1 0 1 1 1 0	E_1 E_2 A 0 0 1 0 1 0 1 0 0 1 1 1

TRUTH TABLE: 4-INPUT 16-OUTPUT DECODER

INPUT	OUTPUTS Y-															
ABCD	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0001	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0010	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0011	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0100	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0101	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0110	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0111	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1000	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1001	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1010	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1011	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1100	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1101	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1110	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Abbildung 2.7
Wahrheitstabelle: Decoder für 4 Ein- und 16 Ausgänge

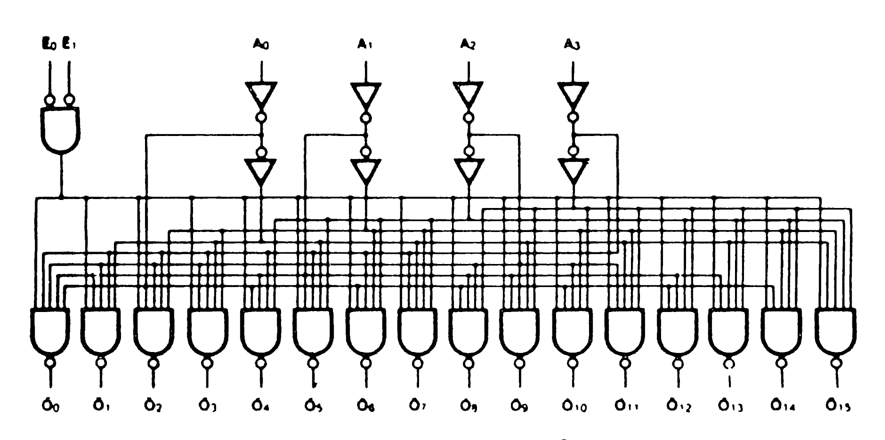


Abbildung 2.8
Decoderschaltung: 4 Ein- und 16 Ausgänge

Decoder wie der von oben werden in der Form eines kleinen Bauteils angeboten. Diese Chips haben 24 Anschlüsse, sogenannte Pins (4 Eingänge, 16 Ausgänge, 2 Stromanschlüsse und 2 "Enable"-Eingänge). Damit überhaupt ein Ausgang "wahr" führen kann, müssen diese beiden Enable-Eingänge "wahr" zeigen. Es gibt auch z. B. 3-8-Decoder oder 2-4-Decoder. Die Ausgänge dieser Bauteile sind oft invertiert im Gegensatz zu den oben genannten; d. h. der angewählte Ausgang führt LOW und alle anderen HIGH.

Abbildung 2.9 zeigt einen typischen TTL-IC mit einigen logischen Gattern.

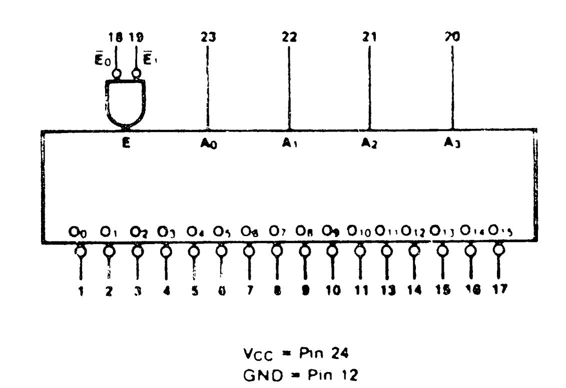


Abbildung 2.9

Decoder und Speicher...

Decoder sind im Computer ein wichtiger Bestandteil bei Operationen mit dem Speicher. Der Speicher besteht aus einer großen Anzahl von Plätzen, auf denen der Computer "1" oder "0" speichern und abfragen kann. In 8-Bit-Computern sind diese Stellen zu Bytes mit je 8 Bit zusammengefaßt. Jedes Byte hat eine bestimmte Adresse. Die Zentraleinheit (CPU) erreicht ein bestimmtes Byte auf folgende Art und Weise:

1.) Die CPU setzt eine Schreib/Lese-Steuerleitung auf den notwendigen Wert (HIGH, LOW), um die benötigte Operation anzuzeigen.

2.) Die CPU gibt die Adresse des gesuchten Bytes aus.

Die Ausgabe erfolgt über bestimmte Leitungen, "Adress-Bus" genannt, in binärer Form. Bei den meisten Kleincomputern werden 16 Adressenleitungen verwendet. Somit können von der CPU 65536 Bytes angesprochen, d.h. 8 * 65536 = 524288 Bits kontrolliert werden. Die CPU müßte also kein 16-65536-Decoder sein. Der Großteil der Decodierung wird aber im Speicher selbst vollzogen, es muß also eine CPU mit über 65000 Pins gebaut werden.

Die 8 Bits einer Adresse werden bei Leseoperationen über 8 Leitungen (Daten-Bus) an die CPU weitergegeben. Bei einer Schreiboperation verläuft der Datentransport in einer anderen Richtung.

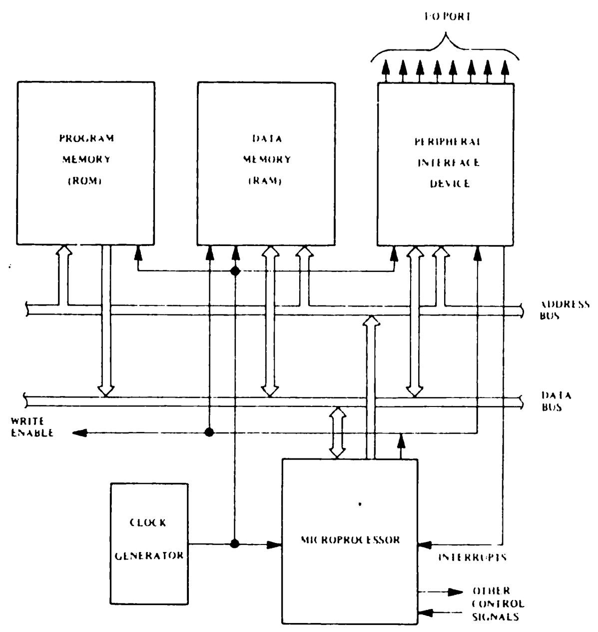


Abbildung 2.10
CPU-Bus-System

NICHT-UND-, WEDER-NOCH-, UND-,EXCLUSIV-ODER-Gatter...

Stellen Sie sich vor, an dem Ausgang eines UND-Gatters wird eine Negationsschaltung angeschlossen. Wir hätten folgende logische Operation (A, B=Eingänge, Z=Ausgang):

Wenn (A=wahr) und (B=wahr), dann (Z=falsch)

Diese logische Funktion nennen wir "NICHT-UND-Gatter". Wir schreiben dafür: Z=A NICHT-UND B. eine zweite Negationsschaltung ergibt wieder eine UND-Schaltung. NICHT-UND-Gatter sind einfacher herzustellen und deshalb sind NICHT-UND-Gatter billiger und mehr im Gebrauch.

Ebenso gibt es ODER-Schaltungen mit invertierten Ausgängen (sog. WEDER-NOCH-Gatter). Diese Gatter sind auch weiter verbreitet als ODER-Gatter. WEDER-NOCH-Gatter werden durch Kreise gekennzeichnet. (Siehe Abbildung 2.11 und 2.12).

Neben dem ODER-Gatter gibt es noch ein EXCLUSIV-ODER-Gatter, bei dem beide Eingänge wahr sein müssen.

Wenn (A=wahr) oder (B=wahr), und (A=falsch) oder (B=falsch), dann (Z=wahr)

Abbildung 2.13 zeigt das logische Symbol dafür.

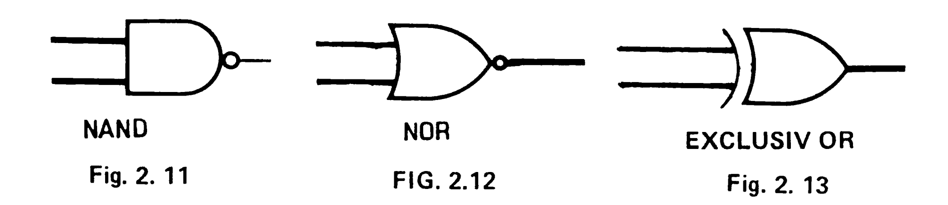


Fig. 2. 11 FIG. 2.12 Fig. 2. 13

Liste der Cursorkontrollzeichen

ASCII	PRINTS	DESCRIPTION	
146		Reverse Aus	Use CTRL key
18	R	Reverse Ein	Use CTRL key
3	C	RUN / STOP	
147	♥	Bildschirm löschen	
19	S	Cursor nach oben links	
145		Cursor nach oben	
17	Q	Cursor nach unten	
157		Cursor nach links	
29	]	Cursor nach rechts	
148		Insert	

Hier noch einige Beispiele:

```
10 PRINT"[illegible]":REM CURSOR HOME
20 PRINT"[illegible]":REM CLEAR HOME
30 PRINT"[illegible]":REM CURSOR NACH RECHTS
40 PRINT"[illegible]":REM CURSOR NACH LINKS
50 PRINT"[illegible]":REM CURSOR NACH UNTEN
60 PRINT"[illegible]":REM CURSOR NACH OBEN
```

Es wurden jeweils fünf der Zeichen nebeneinander eingegeben. Teilweise werden auch die Steuerzeichen für die Farben verwendet.

Das Zeichen DEL ist ähnlich wie Cursor nach links jedoch in die horizontale Achse gespiegelt. Siehe hierzu auch die Liste der Cursorkontrollzeichen.

Maschinensprachen Monitor für den C-64

Maschinensprachen Monitor für den Commodore 64

Das nachfolgende BASIC-Programm erlaubt Ihnen das Experimentieren und Programmieren in Maschinensprache mit Ihrem CBM 64. Nach dem Starten des Monitors können Sie den Inhalt einzelner Speicherzellen ansehen, und wenn gewünscht ändern (S-Funktion).

Sie können einen bestimmten Speicherbereich hexadezimal auf dem Bildschirm auslisten (D = DUMP). Sie können den Inhalt eines Speicherbereiches verschieben (M = Move) und Maschinenprogramme auf Cassette speichern und wieder einlesen. Wichtig ist, daß Sie sich die Startadresse des abzuspeichernden Maschinenpogrammes sowie den Namen des Files merken. Dieser Monitor genügt für die ersten "Ausflüge" in den Speicher des Commodore 64. Sie können ROM und die Adressen in der Zeropage durchsuchen und viele weitere kleine Experimente durchführen.

Für die ersten Gehversuche in der Maschinensprachen-Programmierung ist dieser kleine Monitor auch völlig ausreichend.

Ein einfacher Monitor für den C-64

Nachfolgend finden Sie ein Programm für einen einfachen Monitor für den Maschinensprachenprogrammierer. Das Programm arbeitet auf der Grundversion des C-64.

S = Substitute Speicher. Ändern des Inhaltes einer Speicherzelle.
D = Dump. Ausgabe von Speicherzellen auf dem Bildschirm.
W = SAVE (Speichern von Maschinenprogrammen auf Cassette)
R = Lesen von Maschinenprogrammen auf Cassette
M = Blocktransfer

Der sehr einfache Monitor ist in erster Linie für den ersten Kontakt mit Ihrem C-64 von Commodore gedacht. Er erlaubt dem Anwender kleine Expeditionen im ROM- und RAM-Bereich.

Bei Arbeiten mit dem Monitor hat sich gezeigt, daß man kleine Maschinensprachenexperimente ab der Adresse C000 durchführen kann.

Von 0000 Hex bis ca. 03FB befinden sich die BASIC-Vektoren und Adressen, die der C-64 für seinen eigenen Betrieb benötigt. Von 400 Hex bis 07FF befindet sich der Bildschirmspeicher. Ab 800 hex (2048 dez) beginnt der BASIC-Textbereich. Beim C-64 endet der BASIC-Textbereich bei BFFF hex bzw. CFFF hex. Damit stehen im C-64 ca. 38k RAM ab Adresse 2048 für Anwenderprogramme zur Verfügung. (Ab C000 liegt ein geschützter BASIC-Bereich.)
Das Programm läßt dem Anwender ca. 30k Byte. Mit dem S-Befehl können Sie Speicherzellen ansehen und evtl. ändern. Nach Eingabe der Anfangsadresse können Sie die Adressen durch Drücken der Space-Taste erhöhen und mit der Taste ↑ (Pfeil nach oben) die Adressen erniedrigen.

Wichtig ist, daß Anfangs- und Endadresse immer vierstellig in HEX eingegeben werden. Auch die Ausgabe erfolgt in HEX.

Zur Beendigung der Eingabe mit S drücken Sie Return. Zurück ins Menü immer mit der Spacetaste.

Sie können dann zurück in den Monitor. Erneutes Drücken von RETURN bringt Sie wieder in den BASIC-Interpreter.

Bei all Ihren Maschinensprachenexperimenten mit diesem Monitor müssen Sie also darauf achten, daß Sie nicht mit dem eigentlichen BASIC-Programm des Monitors oder mit dem Bildschirmspeicher in Berührung kommen. Der Bereich um 1C00 Hex dürfte hier ganz gut geeignet sein, um Maschinensprachenprogramme sicher abzulegen.
Der Monitor erlaubt Ihnen weiterhin, Maschinenprogramme auf Cassette abzulegen und wieder einzulesen. Beim Abspeichern (SAVE) muß die Anfangs- und Endadresse sowie ein Name eingegeben werden. Beim Wiedereinladen muß der Name eingegeben werden. Nach der Aufforderung: "REWIND und DRUECKE TASTE", müssen Sie irgendeine Taste auf der C-64-Tastatur drücken. Am Bildschirm erscheint dann: "PRESS PLAY ON TAPE". Sie können jetzt das vorher gespeicherte Programm wieder einladen.

Beschreibung zum Monitor für C-64
Da unser Monitor relativ klein ist, könnte man ihn auch in den geschützten BASIC-Bereich ab Adresse C000 Hex legen und könnte dann seine Maschinensprachenprogramme ab 2048 dez ablegen und hat damit ca. 38k Byte für seine Maschinensprachenexperimente.

Hierzu müssen Sie Ihren C-64 vor dem Laden des Hexmons so präparieren, daß der BASIC-Arbeitsspeicher bei C000 = Hex (49152 dez) beginnt.

```
POKE 44,192
POKE 55,240
POKE 56,207
POKE 642,192
POKE 643,240
POKE 644,207
POKE 49152,0
NEW
```

Hexmonitor laden und mit RUN starten. Wenn Sie jetzt die Pointer wieder zurücksetzen, können Sie ein BASIC-Programm an die Adresse 2048 laden (Standard) und nach Umschalten auf den Bereich C000 mit dem dort liegenden Monitor das BASIC-Programm untersuchen und verändern.

Zurücksetzen der Pointer:

```
POKE 43,0:POKE 44,8:POKE 2048,0:NEW
```

auf C000 setzen:

```
POKE 43,0:POKE 44,192:POKE 49152,0
```

```
8003 PRINT"COPYRIGHT 1983 BY ING.W.HOFACKER GM
BH"
8006 PRINT"ALLE ZAHLEN BITTE IN"
8007 PRINT"HEX UND ADRESSEN IMMER 4STELLEN EIN
-     SCHL.NULLEN V.D.K)"
8008 PRINT"EINZELNE BUCHSTABEN SPEZIFIZIEREN D
IE    BEFEHLE"
8010 REM RESTART
8015 PRINT
8020 PRINT"S=SUBSTITUTE SPEICHER"
8030 PRINT"D=DUMP SPEICHERINHALT"
8040 PRINT"G=GO STARTE EIN PROGR.A.ADD."
8050 PRINT"M=MOVE BLOCKVERSCHIEBUNG"
8060 PRINT"W=WRITE SCHREIBE AUF CASS"
8070 PRINT"R=READ LESE VON CASSETTE"
8080 PRINT"RETURN=RETURN NACH BASIC"
8090 PRINT
8091 PRINT"WENN DAS PROMT * ERSCHEINT, BEFEHL (1
     BUCHSTABE) EINGEBEN"
8095 PRINT
8096 PRINT"* ";
8110 GET C$:IF C$="" GOTO 8110
8120 PRINTC$
8130 IF C$="S" GOTO 8400
8140 IF C$="D" GOTO 8700
8150 IF C$="G" GOTO 9300
8160 IF C$="M" GOTO 9400
8170 IF C$="W" GOTO 9700
8180 IF C$="R" GOTO 9900
8190 IF ASC(C$)=13 THEN END
8200 PRINT"UNGUELTIGER BEFEHL"
8210 PRINT
8220 PRINT
8230 GOTO 8010
8400 PRINT"NAECHSTE ADRESSE ERREICHEN SIE DURCH
     SPACE ZURUECK MIT ↑"
8403 PRINT"AENDERN DES SPEICHERINHALTES DURCH EI
N- GABE DER ZWEISTELLIGEN";
8404 PRINT"  HEXZAHL-MIT      RETURN VERLASSEN S
IE DEN  MODUS WIEDER"
8405 PRINT
8410 PRINT"ANFANGSADRESSE";
8420 INPUT HEX4$
```

```
8430 GOSUB 9000
8440 S=DEC4+1
8450 PRINT
8460 S=S-1
8470 GOSUB 9100
8480 PRINT"  ";
8490 S0=PEEK(S)
8500 N=3:GOSUB 9150
8510 PRINT" - ";
8520 DEC2=0
8530 FOR I=2 TO 1 STEP-1
8540 GET C$:IF C$="" GOTO 8540
8550 IF C$="↑" GOTO 8450
8560 IF C$=" " GOTO 8640
8570 TEMP=ASC(C$)
8580 IF TEMP=13 GOTO 8095
8590 PRINTC$;
8600 F=48:IF TEMP>58 THEN F=55
8610 DEC2=DEC2+(TEMP-F)*I*I*I*I
8620 NEXT I
8630 POKE S,DEC2
8640 S=S+1
8650 PRINT
8660 GOTO 8470
8700 REM HEX DATEN INS RAM    M
8705 PRINT
8710 PRINT"ANFANGSADRESSE";
8720 INPUT HEX4$
8730 GOSUB 9000
8740 L=DEC4
8745 PRINT
8750 PRINT"ENDADRESSE";
8760 INPUT HEX4$
8765 PRINT
8770 GOSUB 9000
8780 H=DEC4
8790 D=H-L
8800 FOR I=1 TO D STEP 8
8810 S=L-1+I
8820 GOSUB 9100
8830 PRINT"   ";
8840 FOR J=1 TO 8
8850 S0=PEEK(S-1+J)
```

```
8860 N=3:GOSUB 9150
8870 PRINT"  ";
8880 NEXT J
8890 PRINT" "
8900 NEXT I
8910 GOTO 8095
9000 REM HEX STRING IN DEZIMAL
9010 DEC4=0:MULT=4096
9020 FOR I=1 TO 4
9030 TEMP=ASC(MID$(HEX4$,I,1))
9040 F=48:IF TEMP>58 THEN F=55
9050 DEC4=DEC4+(TEMP-F)*MULT
9060 MULT=MULT/16
9070 NEXT I
9080 RETURN
9100 REM GIBT ZWEI ODER 4 HEX ZEICHEN
 AUS
9110 N=1
9120 S(1)=INT(S/4096)
9130 S(2)=INT((S-S(1)*4096)/256)
9140 S0=(S-S(1)*4096)-S(2)*256
9150 S(3)=INT(S0/16)
9160 S(4)=(S0-S(3)*16)
9170 FOR K=N TO 4
9180 F=48:IF S(K)>9 THEN F=55
9190 PRINTCHR$(S(K)+F);
9200 NEXT K
9210 RETURN
9300 REM MASCHINENSPRACHENPROGR.
9305 PRINT
9310 PRINT"STARTADRESSE";
9320 INPUT HEX4$
9330 GOSUB 9000
9340 SYS(DEC4)
9350 GOTO 8095
9400 REM VERSCHIEBE DATEN
9405 PRINT
9410 PRINT"STARTADRESSE";
9420 INPUT HEX4$
9430 GOSUB 9000
9440 L=DEC4
9445 PRINT
9450 PRINT"ENDADRESSE";
```

```
9460 INPUT HEX4$
9470 GOSUB 9000
9480 H=DEC4
9485 PRINT
9490 PRINT"NEUE ANFANGSADRESSE";
9500 INPUT HEX4$
9510 GOSUB 9000
9520 E=DEC4
9530 D=H-L
9540 FOR I=0 TO D
9550 J=I:IF L<E THEN J=D-I
9560 S=PEEK(L+J)
9570 Y=E+J
9580 POKE Y,S
9590 NEXT I
9600 GOTO 8095
9700 REM SCHREIBT AUF CASS
9705 PRINT
9710 PRINT"STARTADRESSE";
9720 INPUT HEX4$
9730 GOSUB 9000
9740 L=DEC4
9745 PRINT
9750 PRINT"ENDADRESSE";
9760 INPUT HEX4$
9770 GOSUB 9000
9780 H=DEC4
9790 D=H-L
9791 PRINT
9792 FS=-1:FD=-1:FC=191
9793 PRINT"FILENAME";
9794 INPUT F$
9795 OPEN1,1,1,F$
9797 J=LEN(STR$(D))
9799 FS=FS+J
9810 PRINT#1,D
9820 FOR J=0 TO D
9830 I=PEEK(L+J)
9840 PRINT#1,I
9841 K=LEN(STR$(I))
9842 FE=INT(-(FS+K)/FC)
9843 IF FE=FD GOTO 9849
9845 T=TI
```

```
9846 IF(TI-T)<5 GOTO 9846
9848 FS=-1
9849 FS=FS+K
9850 NEXT J
9860 CLOSE 1
9865 PRINT
9870 PRINT"DATEN AUF BAND GESPEICHERT"
9880 GOTO 8095
9900 REM LIEST DATEN VON CASS
9905 PRINT
9906 PRINT"FILENAME";
9907 INPUT F$
9908 PRINT
9910 PRINT"BAND ZURUECKSPULEN UND BELIEBIGE TAST
E  DRUECKEN!!"
9920 GET A$:IF A$=""THEN 9920
9930 OPEN1,1,0,F$
9940 INPUT#1,D
9945 PRINT"⊐"
9950 PRINT"STARTADRESSE";
9960 INPUT HEX4$
9970 GOSUB 9000
9974 PRINT
9975 L=DEC4
9980 FOR I=0 TO D
9990 INPUT#1,X
9991 PRINT"X";
9992 POKE L+I,X
9993 NEXT I
9994 CLOSE1
9995 PRINT:PRINT
9996 PRINT"ENDE DES FILES"
9997 GOTO8095

READY.
```

Ein Miniassembler für C-64

Ein Miniassembler für C-64

Dieses Programm dient der schnellen und einfachen Erstellung von Maschinenprogrammen auf dem C-64.

Nach dem Programmstart und der Entscheidung für hexadezimale oder dezimale Eingabe, wird nach der Anfangsadresse gefragt. Sie sollte über 2000 Hex (8192 dez) liegen, damit der Assembler selbst nicht zerstört wird. Nach oben hin ist die Anfangsadresse durch die Verwendung der Integervariablen begrenzt. Es sollte also nicht mehr als 7000 oder 28672 dez eingegeben werden.

Der vor BASIC geschützte Bereich ab 033C hex (Cassettenpuffer) kann auch für Maschinenprogramme verwendet werden.

Cassettenpuffer des C-64 033C – 03FB hex
828 – 1019 dez

Man kann im Programm wählen, ob die Eingabe in Hexadezimal oder in Dezimal erfolgen soll. Hat man sich für eine Eingabeart entschieden, so erwartet das Programm, daß sämtliche Eingaben in die gewählte Art gemacht werden, bzw. eine Modusänderung erfolgt.

Das Programm stellt drei Auswahlkriterien zur Verfügung:

1. Modusänderung
Man kann jederzeit im Programm den Modus ändern, d. h. von Dezimal auf Hex oder umgekehrt übergehen. Bei der Eingabe von Adressen zeigt das Programm an, in welchem Modus es die Eingabe erwartet.

2. List
Bei Wahl der Funktion List stellt das Programm 4 Möglichkeiten zur

Verfügung, in denen es ausdrucken kann:

a) DEZ–ASS : Adressen dezimal, Befehle in Assembler
b) HEX–ASS : Adressen hexadezimal, Befehle in Assembler
c) DEZ–DEZ : Adressen und Befehle dezimal
d) HEX–HEX : Adressen und Befehle hexadezimal.

Der Ausdruck erfolgt wahlweise auf Drucker oder Bildschirm. Der Ausdruck NIO bedeutet, daß eine nicht identifizierbare Operation gefunden wurde.

3. Programm
Programm bedeutet, daß jetzt ein Programm geschrieben wird. Dazu wird der Adressenanfang im vorher gewählten Modus eingegeben.

Auf dem Bildschirm wird die erste Adresse ausgedruckt und das Programm erwartet jetzt die Eingabe eines Assemblerbefehls. Die Eingabe wird durch Drücken der Taste "Return" abgeschlossen. Das Programm erhöht die Adresse automatisch um die erforderliche Zahl.

Vorwärts- und Rückwärtsverzweigungen werden automatisch berechnet. Es können vom Verzweigungsbefehl aus auch Adressen angesprungen werden, die weiter als 127 Adressen vor- oder zurück liegen. Das Programm schreibt sich dafür automatisch ein Programm, das den Sprung dann ausführt.

Auf der rechten Seite des Bildschirms werden die Adressen und die jeweiligen Werte, die darin gespeichert werden, zur Kontrolle ausgedruckt.

Bei fehlerhaften Eingaben erfolgen die Fehlermeldungen "Befehl nicht vorhanden" falls ein Befehl erwartet wurde und "unzulänglicher Wert", falls z. B. eine Zahl größer als 255 in den Akku geladen werden soll.

Die Eingabe der Befehle erfolgt so, daß unmittelbar an den Befehl die Adressierungsart angehängt wird.

Hierbei bedeutet:

AB = absolute
IM = immediate
AX = absolute + Indexregister X
AY = absolute + Indexregister Y
IX = indirekt + Indexregister X
IY = indirekt + Indexregister Y
ZP = Zeropage
ZX = Zeropage + Indexregister X
ZY = Zeropage + Indexregister Y

LDAAB heißt also, daß der Akku mit dem
8192 Inhalt der Adresse 8192 geladen
werden soll.

Man kann im Programm Masken setzen, die von jeder Stelle des Programms aus angesprungen werden können. Für diese Masken können die Buchstaben G – Z verwendet werden. A – F sind nicht zulässig, da bei Verwendung dieser Buchstaben eine eindeutige Unterscheidung zwischen Label und Hexzahl nicht mehr möglich wäre.

Bei Wahl einer falschen Marke erfolgt die Fehlermeldung "Label nicht verfügbar". Eine Marke wird gesetzt, indem man den Buchstaben und dahinter einen Punkt schreibt.

Z. B. 8192 G. bedeutet, daß in Adresse 8192 ein Label gesetzt wurde.

8192 JMPAB bedeutet, daß absolut auf Maske
6901 G G gesprungen werden soll.

KOR Durch Eintippen von KOR anstelle eines Befehls kann man den Adressenanfang neu festlegen.

INF Durch Eintippen von INF anstelle eines Befehls springt man aus dem Programm. Die absoluten Werte des Labels werden hierbei in das Maschinenprogramm eingesetzt und das Programm stellt wieder die 3 Möglichkeiten LIST, MODUS und PROGRAMM zur Verfügung.

Während der Programmentwicklung kann man das erstellte Programm zum Testen über SYS (Adresse in dezimal) aufrufen.

Programmbeispiel:
Das folgende Maschinenprogramm läßt einen Asterisk durch Drücken der Taste "⟩" über eine Bildschirmzeile wandern. Durch Betätigen der Taste "⟨" Rückkehr aus dem Maschinenprogramm.

So steht das Programm auf dem Bildschirm, nachdem Sie es eingetippt haben.

```
828 K.
829 LDXIM
830 0
831 H.
832 LDYZP
833 197
834 CPYIM
835 44
836 BNE
837 G
838 RTS
839 G.
840 CPYIM
841 47
842 BNE
843 H
844 LDAIM
845 32
846 STAAX
847 1504
849 INX
850 LDAIM
851 1
852 STAAX
853 55776
855 LDAIM
856 42
857 STAAX
858 1504
860 LDYIM
861 0
862 I.
863 INY
864 CPYIM
865 255
866 BNE
867 I
868 CPXIM
869 39
870 BNE
871 H
872 JMPAB
873 K
```

So sieht das Programm nach der Labelübersetzung aus, wenn bei LIST die Variante DEZ-ASS gewählt wurde.

```
828 NOP            852 STAAX
829 LDXIM          853 55776
830 0
831 NOP            855 LDAIM
832 LDYZP          856 42
833 197            857 STAAX
834 CPYIM          858 1504
835 44
836 BNE            860 LDYIM
837 840            861 0
838 RTS            862 NOP
839 NOP            863 INY
840 CPYIM          864 CPYIM
841 47             865 255
842 BNE            866 BNE
843 832            867 863
844 LDAIM          868 CPXIM
845 32             869 39
846 STAAX          870 BNE
847 1504           871 832
                   872 JMPAB
849 INX            873 829
850 LDAIM
851 1              875 BRK
```

NOP Entstehen dadurch, daß beim Schreiben des Programms an diesen Stellen Marken gestanden haben, die nach der Labelübersetzung aber nicht mehr gebraucht wurden.

Und so sieht der Hexdump aus (bei LIST HEX-HEX gewählt).

```
033C 00EA          0342 00C0
033D 00A2          0343 002C
033E 0000          0344 00D0
033F 00EA          0345 0002
0340 00A4          0346 0060
0341 00C5          0347 00EA
```

```
0348 00C0        035A 00E0
0349 002F        035B 0005
034A 00D0        035C 00A0
034B 00F4        035D 0000
034C 00A9        035E 00EA
034D 0020        035F 00C8
034E 009D        0360 00C0
034F 00E0        0361 00FF
0350 0005        0362 00D0
0351 00E8        0363 00FB
0352 00A9        0364 00E0
0353 0001        0365 0027
0354 009D        0366 00D0
0355 00E0        0367 00D8
0356 00D9        0368 004C
0357 00A9        0369 003D
0358 002A        036A 0003
0359 009D        036B 0000
```

Unser kleines Maschinensprachenbeispiel haben wir in den Cassettenpuffer ab 033C hex abgelegt. Dort liegt es geschützt vor BASIC und vor unserem Supermon 64. Mit dem Supermon 64 haben wir nun das Programm noch einmal disassembliert und ein Hexdump erstellt.

```
.M 033C 036F
.:033C EA A2 00 EA A4 C5 C0 2C
.:0344 D0 02 60 EA C0 2F D0 F4
.:034C A9 20 9D E0 05 E8 A9 01
.:0354 9D E0 D9 A9 2A 9D E0 05
.:035C A0 00 EA C8 C0 FF D0 FB
.:0364 E0 27 D0 D8 4C 3D 03 00
```

```
., 033C EA          NOP
., 033D A2 00       LDX #$00
., 033F EA          NOP
., 0340 A4 C5       LDY $C5
., 0342 C0 2C       CPY #$2C
., 0344 D0 02       BNE $0348
., 0346 60          RTS
```

```
., 0347 EA          NOP
., 0348 C0 2F       CPY #$2F
., 034A D0 F4       BNE $0340
., 034C A9 20       LDA #$20
., 034E 9D E0 05    STA $05E0,X
., 0351 E8          INX
., 0352 A9 01       LDA #$01
., 0354 9D E0 D9    STA $D9E0,X
., 0357 A9 2A       LDA #$2A
., 0359 9D E0 05    STA $05E0,X
., 035C A0 00       LDY #$00
., 035E EA          NOP
., 035F C8          INY
., 0360 C0 FF       CPY #$FF

., 0362 D0 FB       BNE $035F
., 0364 E0 27       CPX #$27
., 0366 D0 D8       BNE $0340
., 0368 4C 3D 03    JMP $033D
., 036B 00          BRK
., 036C 00          BRK
```

Programmbeschreibung

Wir laden das X-Register mit 0 und bringen den Inhalt der Zelle $C5 = 197 dez in das Y-Register.

Die Speicherzelle 197 dez enthält das zuletzt gedrückte Zeichen. Durch Probieren haben wir die Werte für die Zeichen in Zelle 197 herausgefunden.

```
10 PRINT PEEK (197)
20 GOTO 10
```

Die Werte für die "<" und ">" Taste ergaben sich hier mit 44 und 47 (2C und 2F hex).
Wenn diese Tasten in unserem Programm gedrückt werden, werden sie im Y-Register übernommen. Mit den CPY Befehlen in 0342 hex und 0348 hex fragen wir den Y-Register ab, ob diese Werte gedrückt wurden.

Wird die "〉" Taste gedrückt, laden wir den Akkumulator mit #$20 (20 hex = 32 dez = Leerzeichen, siehe ASCII und CHR$ Code Tabelle im mitgelieferten Handbuch). Dieses Leerzeichen wird nun in die erste Bildschirmzelle in der Zeile 13 gebracht (siehe Bildschirmadressen im Handbuch 1504 dez = 05E0 hex). Die indizierte Adressierung über X ermöglicht es nun durch Inkrementieren durch INX die Bildschirmadresse um eins zu erhöhen. In den Farbbildschirm laden wir nun die Farbe (schwarz) mit LDA#$01, STA $D9E0,X. Dann wird das Asterisk Zeichen gedruckt.

LDA #$2A, STA $05E0,X (2A hex = *)

Jetzt folgt eine kleine Warteschleife.

Die Schleife besteht aus den Befehlen:

```
035C  LDY #$00
035E  NOP
025F  INY
0360  CPY #$FF
0362  BNE $035F
```

In Zelle 364 hex wird das X-Register abgefragt,ob das Zeilenende (39 dez) bereits erreicht ist.

```
1 PRINT CHR$(5)
2 POKE 53280,0:PRINT"□":PRINT"■■■■ASSEMBLER VON
ELCOMP":PRINTCHR$(144)
3 FOR Q=1 TO 4000:NEXT
4 CLR:Z$(0)="*":Z$(3)="<":Z$(4)=">":MO$(0)="DEZ"
:MO$(1)="HEX"
5 DIMLA$(20):HE$="0123456789ABCDEF"
6 PRINT"□M O D U S ?":PRINT:PRINT"0=DEZ-ASS":PRI
NT"1=HEX-ASS":PRINT
7 INPUTZO%:GOTO1990
10 DATA0000,BRK,1001,ORAIX,1005,ORAZP,1006,ASLZP
,0008,PHP,2009,ORAIM,0010,ASLAC
20 DATA4013,ORAAB,4014,ASLAB,3016,BPL,1017,ORAIY
,1021,ORAZX,1022,ASLZX,0024,CLC
30 DATA4025,ORAAY,4029,ORAAX,4030,ASLAX,4032,JSR
```

```
,1033,ANDIX,1036,BITZP
40 DATA1037,ANDZP,1038,ROLZP,0040,PLP,2041,ANDIM
,0042,ROLAC,4044,BITAB
50 DATA4045,ANDAB,4046,ROLAB,3048,BMI,1049,ANDIY
,1053,ANDZX,1054,ROLZX
60 DATA0056,SEC,4057,ANDAY,4061,ANDAX,4062,ROLAX
,0064,RTI,1065,EORIX,1069,EORZP
70 DATA1070,LSRZP,0072,PHA,2073,EORIM,0074,LSRAC
,4076,JMPAB,4077,EORAB
80 DATA4078,LSRAB,3080,BVC,1081,EORIY,1085,EORZX
,1086,LSRZX,0088,CLI,4089,EORAY
90 DATA4093,EORAX,4094,LSRAX,0096,RTS,1097,ADCIX
,1101,ADCZP,1102,RORZP,0104,PLA
100 DATA2105,ADCIM,0106,RORAC,1108,JMPIN,4109,AD
CAB,4110,RORAB,3112,BVS
110 DATA1113,ADCIY,1117,ADCZX,1118,RORZX,0120,SE
I,4121,ADCAY,4125,ADCAX
120 DATA4126,RORAX,1129,STAIX,1132,STYZP,1133,ST
AZP,1134,STXZP,0136,DEY
130 DATA0138,TXA,4140,STYAB,4141,STAAB,4142,STXA
B,3144,BCC,1145,STAIY
140 DATA1148,STYZX,1149,STAZX,1150,STXZY,0152,TY
A,4153,STAAY,0154,TXS
150 DATA4157,STAAX,2160,LDYIM,1161,LDAIX,2162,LD
XIM,1164,LDYZP,1165,LDAZP
160 DATA1166,LDXZP,0168,TAY,2169,LDAIM,0170,TAX,
4172,LDYAB,4173,LDAAB
170 DATA4174,LDXAB,3176,BCS,1177,LDAIY,1180,LDYZ
X,1181,LDAZX,1182,LDXZY
180 DATA0184,CLV,4185,LDAAY,0186,TSX,4188,LDYAX,
4189,LDAAX,4190,LDXAY
190 DATA2192,CPYIM,1193,CMPIX,1196,CPYZP,1197,CM
PZP,1198,DECZP,0200,INY
200 DATA2201,CMPIM,0202,DEX,4204,CPYAB,4205,CMPA
B,4206,DECAB,3208,BNE
210 DATA1209,CMPIY,1213,CMPZX,1214,DECZX,0216,CL
D,4217,CMPAY,4221,CMPAX
220 DATA4222,DECAX,2224,CPXIM,1225,SBCIX,1228,CP
XZP,1229,SBCZP,1230,INCZP
230 DATA0232,INX,2233,SBCIM,0234,NOP,4236,CPXAB,
4237,SBCAB,4238,INCAB,3240,BEQ
240 DATA1241,SBCIY,1245,SBCZX,1246,INCZX,0248,SE
D,4249,SBCAY,4253,SBCAX
```

```
250 DATA4254,INCAX
1300 FL%=0:PRINT"■■ADRESSENANFANG":PRINT:PRINTMO
$(MO%):INPUTZ$
1305 IFMO%=1THENGOSUB6000:ZZ%=UM1:Z%=ZZ%:PRINT:P
RINT:GOTO1319
1310 ZZ%=VAL(Z$):Z%=ZZ%
1312 PRINT:PRINT
1316 IFMO%=1THENDE=Z%:GOSUB5000:GOTO1319
1318 Z$=STR$(Z%)
1319 PRINTZ$
1320 INPUT"⌉■■■■■";B$
1321 PRINT"⌉                    ":PRINT"⌉"Z$;"■";B$
1322 IFB$="INF"THEN1980
1323 IFB$="KOR"THEN1300
1324 IFRIGHT$(B$,1)<>"."THENGOTO1327
1325 IFASC(B$)>70THENGOSUB3200:II=234:GOTO1430
1326 PRINT"LABEL NICHT VERFUEGBAR!":GOTO1316
1327 IFP%<>0THEN1470
1330 RESTORE:FORI=1TO151:READA$:READC$:IFB$=C$TH
EN1350
1340 NEXT:PRINT"BEFEHL NICHT VERFUEGBAR!!":RESTO
RE:GOTO1316
1350 RESTORE:II=VAL(MID$(A$,2,3))
1360 P%=VAL(MID$(A$,1,1))
1430 GOSUB4000:POKEZ%,II
1440 Z%=Z%+1
1460 GOTO1316
1470 IFMO%=0THEN1474
1471 Z$=B$:GOSUB6000:X=UM1
1472 FORI=1TOLEN(B$):IFASC(MID$(B$,I,1))>70THENI
FLEN(B$)>1THEN1486
1473 NEXTI:GOTO1475
1474 X=VAL(B$)
1475 IFASC(B$)>70THENGOSUB3200:GOTO1490
1476 IFMO%=0THENIFASC(B$)>57THEN1326
1480 IFX>255THENIFP%<3THEN1486
1482 GOTO1490
1486 PRINT"UNZULAESSIGER WERT!":GOTO1316
1490 ONP%GOTO1500,1500,1600,1520
1500 II=X
1510 GOSUB4000:POKEZ%,II:Z%=Z%+1:P%=0:GOTO1316
1520 H%=INT(X/256):II=X-H%*256
1523 GOSUB4000:POKEZ%,II:PRINT
```

```
1530 II=H%;Z%=Z%+1:GOSUB4000:POKEZ%,II:Z%=Z%+1:P
%=0:GOTO1316
1600 IFX=0THEN1510
1605 IFABS(X-Z%)>127THEN1650
1610 II=X-Z%-1:IFII<0THENII=256+II
1620 GOTO1510
1650 II=3:GOSUB4000:POKEZ%,II:Z%=Z%+1:PRINT
1660 II=76:GOSUB4000:POKEZ%,II:Z%=Z%+1:L=Z%+5:M=
INT(L/256):II=L-M*256:PRINT
1665 GOSUB4000:PRINT:POKEZ%,II:Z%=Z%+1:II=M:GOSU
B4000:PRINT:POKEZ%,II:Z%=Z%+1
1670 II=76:GOSUB4000:POKEZ%,II:Z%=Z%+1:P%=2:PRIN
T:GOTO1520
1980 PRINT"LABELUEBERSETZUNG":GOTO2200
1990 MO%=ZO%:PRINT"J"
2000 PRINT"WAEHLE!":PRINT:PRINT"1.LIST":PRINT"2.
MODUSAENDERUNG"
2005 PRINT"3.PROGRAMM"
2010 GETG%:ONG%GOTO2030,6,1300
2020 GOTO2010
2030 PRINT:PRINT:PRINT"DEZ-ASS=1":PRINT"HEX-ASS=
2":PRINT"DEZ-DEZ=3":PRINT"HEX-HEX=4"
2035 FL%=1:INPUTLD%
2040 PRINT:PRINT:INPUT"DRUCKER (J/N)";DR$
2050 PRINT:PRINT"ANFANG,ENDE"
2055 ZO%=MO%:PRINT:PRINTMO$(MO%):IFMO%=0THEN2065

2060 INPUTNZ$,I$:Z$=NZ$:GOSUB6000:Z%=UM1-2:Z$=I$
:GOSUB6000:E%=UM1-1:GOTO2068
2065 INPUTZ%,E%:Z%=Z%-2:E%=E%-1
2068 IFDR$="J"THENOPEN4,4:CMD4
2070 IFLD%>2THEN7000
2075 IFLD%=2THENMO%=1:GOTO2080
2078 MO%=0
2080 GOSUB3000:V%=VAL(MID$(A$,1,1))
2130 GOSUB3000:V%=VAL(MID$(A$,1,1))
2132 IFMO%=0THEN2135
2134 DE=Z%:GOSUB5000:PRINTZ$;"  ";C$:GOTO2137
2135 PRINTZ%;C$
2137 IFZ%<E%THEN2150
2140 PRINT:PRINT"FERTIG,TASTE DRUECKEN!"
2145 GETG$:IFG$=""THEN2145
2146 IFDR$="J"THENCLOSE4,4
```

```
2148 GOTO1990
2150 IFV%=0THEN2130
2160 Z%=Z%+1:IFV%>2THEN2165
2164 II=PEEK(Z%):GOSUB4000:GOTO2130
2165 P1%=PEEK(Z%):IFV%=4THEN2190
2170 IFP1%>127THENII=Z%+P1%-255:GOSUB4000:GOTO21
30
2175 II=P1%+Z%+1:GOSUB4000:GOTO2130
2190 II=P1%+256*PEEK(Z%+1):GOSUB4000:PRINT
2195 Z%=Z%+1:GOTO2130
2200 FORI=1TO20:P%=0:IFLA$(I)=""THENNEXTI:GOTO4
2210 FORJ=1TOLEN(LA$(I))
2220 IFMID$(LA$(I),J,1)=Z$(P%)THEN2225
2222 GOTO2290
2225 AN%(P%)=VAL(RIGHT$(LA$(I),LEN(LA$(I))-J))
2230 ONP%GOTO2290,2290,2250,2260
2240 GOTO2290
2250 L=AN%(0)-AN%(3):IFL<0THENL=256+L
2255 POKEAN%(3),L:GOTO2290
2260 H%=INT((AN%(0)+1)/256):L=AN%(0)+1-H%*256:PO
KEAN%(4),L:POKEAN%(4)+1,H%
2290 NEXTJ
2300 IFP%=0THENP%=3:GOTO2210
2310 IFP%=3THENP%=4:GOTO2210
2320 NEXTI:GOTO4
3000 Z%=Z%+1:RESTORE:PE%=PEEK(Z%)
3010 FORJ=1TO151:READA$:READC$:IFPE%=VAL(MID$(A$
,2,3))THENRETURN
3020 NEXTJ:C$="NIO":A$="0":RETURN
3200 LA%=ASC(LEFT$(B$,1))-70:ZZ$=STR$(Z%):ZZ$=Z$
(P%)+RIGHT$(ZZ$,LEN(ZZ$)-1)
3300 LA$(LA%)=LA$(LA%)+ZZ$:RETURN
4000 IFMO%=1THENDE=Z%:GOSUB5000:NZ$=Z$:DE=II:GOS
UB5000:I$=Z$:GOTO4090
4050 NZ$=STR$(Z%):I$=MID$(STR$(II),2)
4090 IFFL%<>1THENPRINT"[illegible]";
4100 PRINTNZ$;"  ";I$;"[illegible]":RETURN
5000 Z1=3:Z$=""
5010 Z3=INT(DE/16↑Z1):Z$=Z$+MID$(HE$,Z3+1,1)
5020 DE=DE-Z3*16↑Z1
5030 Z1=Z1-1:IFZ1=-1THENRETURN
5040 GOTO5010
6000 UM1=0:Z1=3
```

```
6005 IFLEN(Z$)<4THENZ$="0"+Z$:GOTO6005
6010 FORI=1TO4:FORJ=1TO16
6020 IFMID$(Z$,I,1)=MID$(HE$,J,1)THENUM1=UM1+(J-
1)*16↑Z1
6030 NEXTJ:Z1=Z1-1:NEXTI:RETURN
7000 IFLD%=4THENMO%=1:GOTO7002
7001 MO%=0
7002 Z%=Z%+2
7005 FORI=Z%TOE%+1:II=PEEK(Z%):GOSUB4000:Z%=Z%+1

7010 NEXT:GOTO2140
```

Notizen

Disassembler

Disassembler

Ein einfacher Disassembler in BASIC für den C-64.

```
10 PRINT"  * 6502 DISASSEMBLER COMMODORE 64 *"
15 PRINT"   ================================":PRINT:PRINT

20 PRINT TAB(23)"COPYRIGHT (C)"
25 PRINT TAB(27)"1983 BY"
30 PRINT TAB(20)"ING.W.HOFACKER GMBH"
35 PRINT:PRINT:PRINT:PRINT
40 DIM A$(255),CO$(15),AD$(3),I$(0),Z$(0),P$(0),L$(0),E$(0
),AD(0),ZA(0),N%(0)
45 DIM Q(0),M(0),Z(0),A(0),P%(2)
50 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F:FOR M=0 TO 15:READ
 CO$(M):NEXT
55 FOR M=0 TO 255:READ A$(M):NEXT:FOR M=1017 TO 1023:READ
N%:POKE M,N%:NEXT
60 PRINT"FOLGENDE FUNKTIONEN STEHEN ZUR"
65 PRINT"VERFUEGUNG:"
70 PRINT:PRINT"DISASSEMBLER        : D":PRINT"ERLAEUTERUNG
    : T"
75 PRINT"BEISPIEL            : B":PRINT"ENDE DES PROGRAMMS:
E"
80 PRINT:PRINT"FUNKTION ? "
85 GET E$:IF E$=""GOTO85
90 IF E$="D"THEN PRINT"":GOTO290
95 IF E$="D"THEN PRINT"":GOTO290
100 IF E$="T"GOTO120
105 IF E$="B"GOTO170
110 IF E$="E"THEN PRINT"":END
115 GOTO80
120 PRINT"DIESER DIASSEMBLER IST AUF DEN GE-      SAMMTEN
 ADRESSIERBEREICH"
125 PRINT"ANWENDBAR,ALSO AUCH AUF DEN PROM-       BEREICH.
":PRINT
```

```
130 PRINT"DIE 'ASSEMBLY LANGUAGE FORM' WIRD        DABEI ET
WAS ABGEAENDERT:"
135 PRINT:PRINT TAB(10)"Z  = ZERO PAGE":PRINT TAB(10)"I  =
 INDIRECT"
140 PRINT TAB(10)"ZX = ZERO PAGE,X":PRINT TAB(10)"IX = (IN
DIRECT,X)"
145 PRINT TAB(10)"IY = (INDIRECT),Y":PRINT
150 PRINT"BEI DEN RELATIVEN SPRUNGBEFEHLEN IST    DAS SPRU
NGZIEL (DEZ.) ANGE-"
155 PRINT"GEBEN UND MIT EINEM STERNCHEN (*)        VERSEHEN
.":PRINT
160 PRINT"CODES,DIE KEINEM BEFEHL ENTSPRECHEN,    SIND MIT
 '*' GEKENNZEICHNET."
165 PRINT:GOTO80
170 PRINT"BEISPIEL:":PRINT:PRINT
175 PRINT"     ADRESSE        CODE     OP-     OPER-"
180 PRINT"  DEZI.  HEX.     OP LO HI  CODE    RAND"
185 PRINT" ──────────────┼──────────┼───────────"
190 PRINT"  58443  E44B     20 69 E2  JSR     57961"
195 PRINT"  58446  E44E     C9 1D     CMP #   29"
200 PRINT"  58448  E450     D0 12     BNE     58468*":PRINT
205 PRINT TAB(31)"↑":PRINT:PRINT TAB(30)"MODE":PRINT:PRINT
:PRINT:PRINT:PRINT
210 GOTO80
290 PRINT:INPUT"STARTADRESSE";AD:INPUT"ENDADRESSE  ";ZA:PR
INT
300 A=AD:GOSUB570:I$=A$(Q)
310 GOSUB500:AD$(1)=L$:IF I$="*"THEN GOSUB520:AD$(2)="*":I
$="":GOTO340
320 ON VAL(RIGHT$(I$,1))GOSUB520,530,550
330 I$=MID$(I$,1,LEN(I$)-1):I$=LEFT$(I$,3)+" "+MID$(I$,4)
340 Z$=AD$(1)+" "+AD$(2)+" "+AD$(3):M=256
350 Q=AD/M:N%=Q-256*INT(Q/256):GOSUB510:AD$(2)=L$
360 IF M=1 THEN AD$(0)=AD$(1)+AD$(2):GOTO380
370 AD$(1)=AD$(2):M=1:GOTO350
380 IF LEFT$(I$,1)="B"GOTO410
390 PRINT AD;TAB(8)AD$(0);TAB(15)Z$;TAB(26)I$;TAB(32)P$:IF
(AD+Z)>ZA GOTO80
400 AD=AD+Z:GOTO300
410 IF LEFT$(I$,3)="BIT"OR LEFT$(I$,3)="BRK"GOTO390
420 N%=VAL(P$):IF N%=>128 THEN P$=STR$(AD+Z+N%-256):GOTO44
0
430 P$=STR$(AD+Z+N%)
```

```
440 P$=P$+"*":GOTO390
500 N%=Q
510 L$=CO$(N%/16)+CO$(N% AND 15):RETURN
520 AD$(2)="":AD$(3)="":Z=1:P$="":RETURN
530 A=AD+1:GOSUB570:P$=STR$(Q):GOSUB500:AD$(2)=L$
540 AD$(3)="":Z=2:RETURN
550 FOR M=1 TO 2:A=AD+M:GOSUB570:P%(M)=Q:GOSUB500
560 AD$(M+1)=L$:NEXT:Z=3:P$=STR$(P%(1)+256*P%(2)):RETURN
570 POKE1019,INT(A/256):POKE1018,A-256*INT(A/256):SYS(1017
):Q=PEEK(1016)
580 RETURN
710 DATA BRK1,ORAIX2,*,*,*,ORA2,ASL2,*,PHP1,ORA#2,ASLA1,*,
*,ORA3,ASL3,*,BPL2
720 DATA ORAIY2,*,*,*,ORAZX2,ASLZX2,*
730 DATA CLC1,ORAY3,*,*,*,ORAX3,ASLX3,*,JSR3,ANDIX2,*,*,BI
T2,AND2,ROL2,*
740 DATA PLP1,AND#2,ROLA1,*,BIT3,AND3,ROL3,*,BMI2,ANDIY2,*
,*,*,ANDZX2,ROLZX2,*
750 DATA SEC1,ANDY3,*,*,*,ANDX3,ROLX3,*,RTI1,EORIX2,*,*,*,
EOR2,LSR2,*,PHA1
760 DATA EOR#2,LSRA1,*,JMP3,EOR3,LSR3,*,BVC2,EORIY2,*,*,*,
EORZX2,LSRZX2,*,CLI1
770 DATAEORY3,*,*,*,EORX3,LSRX3,*,RTS1,ADCIX2,*,*,*,ADC2,R
OR2,*,PLA1,ADC#2
780 DATA RORA1,*,JMPI3,ADC3,ROR3,*,BVS2,ADCIY2,*,*,*,ADCZX
2,RORZX2,*,SEI1
790 DATA ADCY3,*,*,*,ADCX3,RORX3,*,*,STAIX2,*,*,STY2,STA2,
STX2,*,DEY1,*,TXA1,*
800 DATA STY3,STA3,STX3,*,BCC2,STAIY2,*,*,STYZX2,STAZX2,ST
XZY2,*,TYA1,STAY3
810 DATA TXS1,*,*,STAX3,*,*,LDY#2,LDAIX2
820 DATA LDX#2,*,LDY2,LDA2,LDX2,*,TAY1,LDA#2,TAX1,*,LDY3,L
DA3,LDX3,*,BCS2
830 DATA LDAIY2,*,*,LDYZX2,LDAZX2,LDXZY2,*,CLV1,LDAY3,TSX1
,*,LDYX3,LDAX3
840 DATA LDXY3,*,CPY#2,CMPIX2,*,*,CPY2,CMP2,DEC2,*,INY1,CM
P#2,DEX1,*,CPY3,CMP3
850 DATA DEC3,*,BNE2,CMPIY2,*,*,*,CMPZX2,DECZX2,*,CLD1,CMP
Y3,*,*,*,CMPX3,DECX3
860 DATA *,CPX#2,SBCXI2,*,*,CPX2,SBC2,INC2,*,INX1,SBC#2,NO
P1,*,CPX3,SBC3,INC3
870 DATA *,BEQ2,SBCIY2,*,*,*,SBCZX2,INCZX2,*,SED1,SBCY3,*,
*,*,SBCX3,INCX3,*
880 DATA 173,0,0,141,248,3,96
```

Notizen



A9 XX

Es wird das Folgebyte in den Akkumulator geladen.

z.B. A9 AA LDA #$AA

Die Hex-Zahl AA wird in Akku geladen.

AD XX XX

Es wird der Inhalt der Speicherzelle in den Akkumulator geladen, der sich aus den Folgebytes ergibt. Zu beachten wäre Lower- und Higherbyte.

z.B. AD 00 17 LDA $1700

Der Inhalt der Adresse 1700 wird in den Akku geladen.

A5 XX

Es wird der Inhalt der Speicherzelle in den Akkumulator geladen, die sich aus dem Folgebyte ergibt. Es ist eine Adresse in der Zeropage.

z.B. A5 0A LDA $0A

Der Inhalt der Adresse 00 0A wird in den Akku geladen.

A1 XX

Es wird der Inhalt des X-Register's zum Folgebyte addiert. Das Ergebnis dieser Addition ist eine Adresse in der Zeropage. Deren Inhalt ist das Lowerbyte und die nächstfolgende Adresse des Higherbyte. Der Inhalt der Adresse, die sich aus Lower- und Higherbyte ergibt, wird in den Akkumulator geladen.

z.B. A1 03 LDA (03,X)

X-Registerinhalt ist z. B. 04; 03 + 04 → 07. Inhalt der Adresse 0007 ist z. B. AA. Inhalt der nächstfolgenden Adresse, also 0008 ist z. B. 01 Lowerbyte ist AA; Higherbyte ist 01. Es wird also der Inhalt der Adresse 01 AA in den Akku gespeichert.

Wichtig:
Es wird kein Überlauf bei der Addition des X-Registerinhalts und des Folgebytes geprüft. D. h. FE + 04 ergibt nicht 01 02 sondern 02. Die 1 wird nicht berücksichtigt.

B1 XX

Es wird der Inhalt der Adresse des Folgebytes zum Y-Registerinhalt dazu addiert. Dies ergibt eine neue Adresse, deren Inhalt in den Akkumulator geladen wird.
Tritt bei der Addition Folgebyte und Y-Registerinhalt, ein Überlauf auf, (Carry-Flag wird gesetzt) so wird der Überlauf zu der nächstfolgenden Adresse im Folgebyte angegeben ist addiert. Dabei ist nun der Wert ohne Überlauf das Lowerbyte und die Addition, Überlauf und Inhalt der nächstfolgenden Adresse, das Higherbyte. Tritt hier ein Überlauf auf, so wird er nicht berücksichtigt. Der Inhalt der Adresse, die sich aus Lower- und Higherbyte ergibt, wird in den Akkumulator geladen.

1. ohne Überlauf

z.B. B1 07 LDA (07),Y

Y-Registerinhalt ist z. B. 04.
Inhalt der Speicherzelle 0007 ist z. B. 01; 01 + 04 → 05
Der Inhalt der Speicherzelle 0005 wird in den Akku geladen.

2. mit Überlauf

z.B. B1 07 LDA (07),Y

Y-Registerinhalt ist z. B. 04; Inhalt der Speicherzelle 0007 ist z. B. FE; FE + 04 → 1. Überlauf 02 d. h. Überlauf hat stattgefunden (Carry-Flag ist gesetzt). 2 ist nun das Lowerbyte und die Addition des Zelleninhaltes der Adresse Folgebyte 07 + 1 also Adresse 0008 und Überlauf 1 ist das Higherbyte. Z. B. Inhalt der Adresse 0008 ist 04. 04 + Überlauf ist nun das Higherbyte. Lower- und Higherbyte ergeben nun die Adresse 0502. Der Inhalt dieser Adresse wird in den Akku geladen.

Bemerkung:
Ist zum Beispiel der Inhalt der Adresse 0008 = FF, wobei die Addition mit dem Überlauf wieder einen Überlauf erzeugen würde, so wird diesmal der neuerlich erzeugte Überlauf nicht

berücksichtigt. Also FF + 1 → 1 00
Hier würde der Inhalt der Adresse 0002 in den Akku geladen.

B5 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Die Addition ist eine neue Adresse, deren Inhalt in den Akkumulator geladen wird.

z.B.   B5 07   LDA 07,X

X-Registerinhalt ist z. B. 01; 07 + 01 → 08. Inhalt der Adresse 0008 ist z. B. FF, wobei FF nun in den Akku geladen wird.

BD XX XX

Es werden die Adresse, die sich aus den 2 Folgebytes ergibt, Lower- und Higherbyte berücksichtigt, und der Inhalt des X-Register's addiert. Die Addition ergibt eine neue Adresse, deren Inhalt in den Akkumulator geladen wird.

z.B.   BD 00 17   LDA $1700,X

X-Registerinhalt ist z. B. 01; 1700 + 01 → 1701. Inhalt der Adresse 1701 wird in Akku geladen.

B9 XX XX

Ist der analoge Befehl zu BD XX XX, nur im Unterschied dazu, wird hier das Y-Register angesprochen.

z.B.   B9 00 17   LDA $1700,Y

LDX

A2 XX

Es wird das Folgebyte in das X-Register geladen.

z.B. A2 F0 LDX $F0

Die Hex-Zahl F0 wird in X-Register geladen.

AE XX XX

Es wird der Inhalt der Speicherzelle geladen, die sich aus den Folgebytes ergibt. Lower- und Higherbytes berücksichtigt.

z.B. AE 00 10 LDX $1000

Der Inhalt der Adresse 1000 wird in X-Register geladen.

A6 XX

Es wird der Inhalt der Speicherzelle in das X-Register geladen, die sich aus dem Folgebyte ergibt. Es ist eine Adresse in der Zeropage.

z.B. A6 00 LDX $10

Der Inhalt der Adresse 0010 wird in X-Register geladen.

B6 XX

Es wird das Folgebyte mit dem Inhalt des Y-Register's addiert. Die Addition ist eine neue Adresse, deren Inhalt in das X-Register geladen wird.

z.B. B6 08 LDX $08,Y

Y-Registerinhalt sei 01; 08 + 01 → 09. Inhalt der Adresse 09 wird in das X-Register geladen.

BE XX XX

Es werden die Adresse, die sich aus den 2 Folgebytes ergibt, Lower- und Higherbyte berücksichtigt, und der Inhalt des Y-Register's addiert. Die Addition ergibt eine neue Adresse deren Inhalt`in das X-Register geladen wird.

z.B. BE 00 09 LDX $900,Y

Y-Registerinhalt sei z. B. 01; 0900 + 01 → 0901. Inhalt der Adresse 0901 wird im X-Register geladen.



A0 XX

Es wird das Folgebyte in das Y-Register geladen.

z.B. A0 EE LDY #$EE

Die Hex-Zahl EE wird in das Y-Register geladen.

AC XX XX

Es wird der Inhalt der Speicherzelle, die durch die Folgebytes angegeben wird, Lower- und Higherbytes berücksichtigt, in das Y-Register geladen.

z.B. AC 00 08 LDY $800

Der Inhalt der Adresse 0800 wird in das Y-Register geladen.

A4 XX

Es wird der Inhalt der Speicherzelle, die durch das Folgebyte angegeben ist, in das Y-Register geladen.

z.B. B4 0F LDY $0F,X

X-Registerinhalt sei z. B. 02; 0F + 02 → 11. Inhalt der Adresse 11 wird in das Y-Register geladen.

BC XX XX

Es werden die Adresse, die sich aus den 2 Folgebytes ergibt, Lower- und Higherbyte berücksichtigt, und der Inhalt des X-Registers addiert. Das Y-Register wird mit dem Inhalt der aus Addition ergebenen Adresse geladen.

z.B. *BC 00 06* *LDY $600,Y*

X-Registerinhalt sei z.B. 04; 0600 + 04 → 0604. Inhalt der Adresse 0604 wird ins Y-Register geladen.

STA

8D XX XX

Der Inhalt des Akkumulator's wird in die Adresse gespeichert, die sich aus den Folgebytes ergeben, Lower- und Higherbyte berücksichtigt.

z.B. *8D 00 05* *STA $500*

Akkuinhalt sei z.B. DD
DD wird in die Adresse 0500 gespeichert.

85 XX

Der Inhalt des Akkumulators wird in die Adresse gespeichert, die sich aus dem Folgebyte ergibt. Die Adresse liegt in der Zeropage.

z.B. *85 07* *STA $07*

Akkuinhalt sei z. B. 01
01 wird in die Adresse 07 gespeichert.

81 XX

Es wird der Inhalt des X-Register's zum Folgebyte addiert. Das Ergebnis dieser Addition ist eine Adresse in der Zeropage. Deren Inhalt ist das Lowerbyte. Die nächstfolgende Adresse ist das Higherbyte. Der Akkuinhalt wird in die Adresse gespeichert, die sich aus Lower- und Higherbyte ergibt.

z.B. 81 07 STA (07,X)

X-Registerinhalt sei z. B. 04; 07 + 04 → 0B. Inhalt der Adresse 000B sei z. B. AB. Inhalt der nächstfolgenden Adresse also 000C, sei z. B. 02. Lowerbyte = AB; Higherbyte = 02. Es wird der Akkuinhalt in die Adresse 02AB gespeichert.

Bemerkung:
Tritt ein Überlauf bei der Addition des X-Registerinhaltes und dem Folgebyte auf, so wird der Überlauf nicht berücksichtigt. D. h. FF + 02 ergibt 101 wobei nur 01 bewertet wird.

91 XX

Es wird der Inhalt der Adresse des Folgebytes zum Y-Registerinhalt dazu addiert. Diese Addition ergibt eine neue Adresse, in die der Akkuinhalt gespeichert wird.
Tritt bei der Addition zu Folgebyte und Y-Registerinhalt ein Überlauf auf (Carry-Flag wird gesetzt), so wird der Überlauf zu der nächstfolgenden Adresse im Folgebyte angegeben ist, addiert. Der Wert ohne Überlauf ist das Lowerbyte und die Addition plus Überlauf der nächstfolgende Adresseninhalt des Higherbytes. Tritt hier ein Überlauf auf, so wird er nicht berücksichtigt. Der Akkuinhalt wird in die Adresse gespeichert, die sich aus Lower- und Higherbyte ergibt.

1. ohne Überlauf:

z.B. 91 08 STA (08),Y

Y-Registerinhalt 05. Inhalt der Speicherzelle 0008 ist z.B. 04; 04 + 05 → 09. Akkuinhalt wird in Speicherzelle 0009 geladen.

2. mit Überlauf:

z.B. 91 08 STA (08),Y

Y-Registerinhalt 05. Inhalt der Speicherzelle 0008 ist z. B. FD; FD + 05 → Überlauf 1 02. Ein Überlauf hat stattgefunden. 02 ist nun das Lowerbyte. Die Addition des Zelleninhaltes der Adresse Folgebyte + 1 (08 + 1) also Inhalt der Adresse 09 und Überlauf 1 ergeben das Higherbyte. Z.B. Inhalt der Adresse 0009 ist 04. 04 + 1 (Überlauf) ist das Higherbyte also 05 Lower- und Higherbyte ergeben die Adresse 0502. Der Akkuinhalt wird in diese Adresse geladen.

Bemerkung:
Ist z. B. der Inhalt der Adresse 0009 = FF wobei die Addition mit dem vorherigen Überlauf wieder einen Überlauf ergibt, so wird der neuerliche Überlauf nicht berücksichtigt.
Also FF + 1 → 1 00. D. h. 00 ist das Higherbyte. Der Inhalt des Akku würde also in Adresse 00 02 geladen.

95 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Das Ergebnis ist eine neue Adresse, in die der Inhalt des Akkumulator's geladen wird.

z.B.   95 09   STA $09,X

X-Registerinhalt sei z. B. 01; 09 + 01 → 0A. Akkuinhalt wird in Adresse 000A gespeichert.

9D XX XX

Es wird die Adresse, die sich aus den beiden Folgebytes ergibt, (Lower- und Higherbyte) und der Inhalt des X-Register's addiert. Das Ergebnis der Addition ist die Adresse, in die der Akkuinhalt gespeichert wird.

z.B.   9D 00 01   STA $100,X

X-Registerinhalt sei z. B. 01; 0100 + 01 → 0101. Inhalt des Akku wird in Adresse 0101 geladen.

99 XX XX

Dies ist der analoge Befehl zu 9D XX XX, nur daß hier mit dem Y-Register gearbeitet wird.

z.B.   99 00 01   STA $100,Y

STX

8E XX XX

Es wird der Inhalt des X-Registers in die Speicherzelle geladen, die sich aus den beiden Folgebytes ergibt (Lower- und Higherbyte berücksichtigt).

z.B. 8E 00 01 STX $100

X-Registerinhalt wird in Speicherzelle 0100 gespeichert.

86 XX

Es wird der Inhalt des X-Registers in die Speicherzelle geladen, die durch das Folgebyte angegeben ist.
Es ist eine Adresse aus der Zeropage.

z.B. 86 0F STX $0F

X-Registerinhalt wird in der Adresse 000F abgespeichert.

96 XX

Es wird das Folgebyte mit dem Inhalt des Y-Registers addiert. Das Ergebnis ist die Adresse, in das der X-Registerinhalt gespeichert wird.

z.B. 96 F0 STX $F0,Y

Y-Registerinhalt soll z. B. 01 sein. F0 + 01 → F1. Inhalt des X-Registers wird in F1 gespeichert. Kommt bei der Addition ein Überlauf zustande, so wird er nicht verarbeitet.
Z.B. F1 + 0F → 100; hier wird X-Registerinhalt in 0000 gespeichert.

STY

8C XX XX

Es wird der Inhalt des Y-Registers in die Adresse geladen, die sich aus den Folgebytes ergeben (Lower- und Higherbyte berücksichtigt).

z.B. 8C 00 03 STY $300

D. h. Y-Registerinhalt wird in Adresse 0300 abgespeichert.

84 XX

Es wird der Inhalt des Y-Registers in die Adresse geladen, die sich aus den Folgebyte ergibt.

z.B. 84 01 STY 01

D. h. Y-Registerinhalt wird in Adresse 0001 gespeichert.

94 XX

Es wird das Folgebyte mit dem Inhalt des X-Registers addiert. Das Ergebnis ist die Adresse, in die das Y-Register gespeichert wird.

z.B. 94 FF STY $FF,X

X-Registerinhalt sei 02; FF + 02 → Überlauf 101. Y-Registerinhalt wird in Speicherzelle 0001 abgespeichert. Der Überlauf wird nicht verarbeitet. Er geht verloren.

TAX

AA

Der Inhalt des Akkumulator's wird in das X-Register geladen.

z.B. Akkuinhalt ist FF
Nach Ausführung des Befehls steht FF im X-Register.

TAY

A8

Der Inhalt des Akkumulator's wird in das Y-Register geladen.

z.B. Akkuinhalt ist 11
Nach Ausführung des Befehl steht 11 im Y-Register.

TSX

BA

Der Stackpointerinhalt wird in das X-Register geladen.

z.B. Stackpointerinhalt ist 02
Nach Ausführung des Befehls steht 02 im X-Register.

TXA

8A

X-Registerinhalt wird in Akku geladen.

z.B. *X-Registerinhalt ist FA*
Nach der Ausführung des Befehls steht FA im Akku.

TXS

9A

X-Registerinhalt wird in Stackpointeradresse gespeichert.

z.B. *X-Registerinhalt 03*
Nach der Ausführung steht 03 in Stackpointeradresse.

TYA

98

Y-Registerinhalt wird in Akku abgespeichert.

z.B. *Y-Registerinhalt 09*
Nach Ausführung steht 09 im Akku.



69 XX

A + M + C → A

Es wird das Folgebyte und der momentane Akkuinhalt addiert und in Akku abgespeichert. Tritt ein Überlauf auf, so wird der Überlauf mit addiert, wenn vorher Carry-Flag gesetzt war, sonst nicht.

z.B. 69 05 ADC #$05

Akkuinhalt ist FE. FE + 05 → 1 Überlauf 03 + Überlauf 01, und wenn vorher Carry-Flag gesetzt → 04. D. h. 04 wird im Akku abgespeichert.

6D XX XX

Es wird der Inhalt der Adresse die aus den beiden Folgebytes hervor gehen mit dem momentanen Akkuinhalt addiert.
Tritt dabei ein Überlauf auf, so wird dieser dazu addiert, wenn vorher das Carry-Flag gesetzt war; sonst nicht.

z.B. 6D 00 01 ADC $100

Akkuinhalt 0F. Inhalt der Zelle 0100 soll 03 sein.
D. h. 0F + 03 = 12 wird in Akku abgespeichert.

65 XX

Es wird der Inhalt der Adresse, die aus dem Folgebyte hervorgeht mit dem momentanen Akkuinhalt addiert.
Tritt dabei ein Überlauf auf, wird er addiert, wenn vorher das Carry-Flag gesetzt war. War es vorher nicht gesetzt, so wird der Überlauf nicht addiert, sondern nur das Carry-Flag gesetzt.

z.B. 65 03 ADC $03

Akkuinhalt = 04. Inhalt der Zelle 0003 sei 01.
D. h. 01 + 04 → 05 wird in Akku geladen.

61 XX

Es wird der Inhalt des X-Registers zum Folgebyte addiert. Das Ergebnis dieser Addition ist eine Adresse in der Zeropage. Deren Inhalt ist das Lowerbyte und die nächstfolgende Adresse ist das Higherbyte. Der Inhalt der sich aus Lower- und Higherbyte ergibt und der momentane Akkuinhalt wird in den Akku geladen.

z.B. 61 03 ADC (03,X)

X-Registerinhalt ist z. B. 04, momentaner Akkuinhalt ist 06 03 + 04 → 07; Inhalt der Adresse 0007 ist z. B. AA. Inhalt der nächstfolgenden Adresse, also 0008, ist z. B. 01.
Es wird also der Inhalt von der Adresse 01 AA zum momentanen Akkuinhalt dazuaddiert und im Akku gespeichert.

Bemerkung:
Tritt bei der Addition des X-Registerinhaltes und des Folgebytes ein Überlauf auf so wird er dazugezählt, wenn vorher das Carry-Flag gesetzt war. War es nicht gesetzt, so wird der Überlauf vernachlässigt und gleichzeitig wird das Carry-Flag gesetzt.

71 XX

Es wird der Inhalt der Adresse des Folgebytes zum Y-Registerinhalt dazu addiert. Dies ergibt eine neue Adresse, deren Inhalt mit dem momentanen Inhalt des Akku addiert wird und in den Akku gespeichert wird. War vorher noch das Carry-Flag gesetzt, so wird auch dieses subtrahiert. Tritt ein Überlauf in der Addition des Folgebytes mit dem Y-Registerinhalt auf, so wird der Überlauf zu der nächstfolgenden Adresse, die im Folgebyte angegeben ist, addiert. Dabei ist nun der Wert ohne Überlauf das Lowerbyte und die Addition Überlauf und Inhalt der nächstfolgenden Adresse das Higherbyte. Tritt hier ein Überlauf auf, so wird er nicht berücksichtigt. Der Inhalt der Adresse die sich aus Lower- und Higherbyte ergibt wird mit dem momentanen Akkumulatorinhalt addiert. Das Carry-Flag wird vom Ergebnis auch noch subtrahiert.

Ohne Überlauf:

z.B. 71 07 ADC (07),Y

Y-Registerinhalt ist z. B. 04. Inhalt der Speicherzelle 0007 ist

z. B. 01; 01 + 04 → 05. Der Inhalt der Speicherzelle 0005 wird zum momentanen Akkuinhalt addiert. Inhalt sei z. B. 05, angenommen momentaner Akkuinhalt sei 0A und Carry-Flag sei 0. Dann steht nach Befehlsausführung 0F im Akku.

Mit Überlauf:

z.B. 71 07 ADC (07),Y

Y-Registerinhalt sei z. B. 04. Inhalt der Speicherzelle 0007 ist z. B. FE; FE + 04 → 1 (Überlauf) 02. 03 ist nun das Lowerbyte. Die Adresse Folgebyte +1 also 0008 ist z. B. 04. Der Überlauf dazu addiert ergibt: 04 + 1 → 05; 05 ist nun das Higherbyte. Lower- und Higherbyte ergeben nun die Adresse 0502. Der Inhalt dieser Adresse wird mit dem Akkuinhalt addiert. Z. B. momentaner Akkuinhalt sei z. B. 0F
Inhalt der Adresse 0502 z. B. 03.
Nach Befehlsausführung wird 12 als Akkuinhalt stehen. War vor der Befehlsausführung das Carry-Flag gesetzt, so steht nach der Befehlsausführung 13 im Akku.

75 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Die Addition ist eine neue Adresse, deren Inhalt mit dem Akkuinhalt addiert wird. War vor der Befehlsausführung das Carry-Flag gesetzt, so wird auch dieses noch subtrahiert.

z.B. 75 07 ADC 07,X

X-Registerinhalt sei z. B. 02, 07 + 02 → 09. Inhalt der Adresse 0009 sei z. B. FE. Momentaner Akkuinhalt soll z. B. 03 sein. Carry-Flag soll vorher schon gesetzt sein. Nach der Befehlsausführung wird FE + 03 + 01 also 02 im Akku stehen. Der Überlauf wird nicht berücksichtigt in der Rechnung. Das Carry-Flag bleibt daher gesetzt.

7D XX XX

Es wird die Adresse, die sich aus den 2 Folgebytes ergibt (Lower- und Higherbyte) und der Inhalt des X-Register's addiert. Dieses Additionsergebnis ergibt eine neue Adresse, deren Inhalt mit dem Akkumulator

addiert wird. Ein eventuell vorher gesetztes Carry-Flag wird auch noch addiert.

z.B. 7D 01 03 ADC $301,X

X-Registerinhalt z. B. 02. 0301 + 02 → 0303. Inhalt der Adresse 0303 sei z. B. 3F. Inhalt des Akku sei z. B. 05. Carry-Flag sei 0. Nach der Befehlsausführung wird 44 im Akku stehen.

79 XX XX

Ist der analoge Befehl zu 7D XX XX hier wird nur das Y-Register angesprochen.

z.B. 79 01 03 ADC $301,Y

AND

29 XX

UND FUNKTION

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Es wird das Folgebyte mit dem Akkumulatorinhalt logisch durch UND verknüpft. Das Ergebnis steht dann im Akku.

z.B. 29 FF AND #$FE

Akkuinhalt sei z. B. 81

1	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1

1 0 0 0 0 0 0 0

Nach Befehlsausführung wird im Akku 80 als Inhalt sein.

2D XX XX

Der Inhalt der Speicherzelle, die sich aus den beiden Folgebytes ergibt (Lower- und Higherbyte) wird mit dem Inhalt des Akkumulator's logisch UND verknüpft. Das Ergebnis steht nach Befehlsausführung im Akku.

z.B. 2D 01 02 AND $201

Der Inhalt der Speicherzelle 0201 sei 01 angenommen, der Akkuinhalt soll 02 sein. Nach der Logisch UND-Verknüpfung wird 00 im Akku stehen.

25 XX

Der Inhalt der Speicherzelle, die durch das Folgebyte angegeben ist (Zeropageadresse) wird mit Akkumulatorinhalt logisch UND verknüpft. Das Ergebnis steht dann im Akku.

z.B. 25 04 AND 04

Akkuinhalt sei FF. Inhalt der Speicherzelle 0004 sei 01. Nach der Ausführung des Befehls steht 01 im Akku.

21 XX

Es wird der Inhalt des X-Registers zum Folgebyte addiert. Das Additionsergebnis ist eine Adresse in der Zeropage. Deren Inhalt ist das Lowerbyte und der nächstfolgende Adresseninhalt ist das Higherbyte. Der Inhalt der aus Lower- und Higherbyte ergebenden Adresse wird mit dem Akkumulatorinhalt logisch UND verknüpft. Ein eventueller Überlauf bei der Addition wird nicht berücksichtigt.

z.B. 21 05 AND (05,X)

X-Registerinhalt sei 08. 05 + 08 → 0D
Inhalt der Adresse 0D sei z. B 03; Inhalt der Adresse 0E sei z. B. 02; Akkuinhalt sei z. B.. 01. Inhalt der Adresse 0203 sei 02. Nach Einführung des Befehls steht im Akku 00.

31 XX

Es wird der Inhalt der Adresse des Folgebytes zum Y-Register addiert. Das Additionsergebnis ist eine Adresse, deren Inhalt mit dem Akku-

inhalt logisch UND verknüpft wird. Das Ergebnis steht anschließend im Akku. Tritt bei der Addition ein Überlauf auf, d. h. das Carry-Flag wird gesetzt, so wird der Überlauf zu der nächstfolgenden Adresse die im Folgebyte angegeben ist, addiert. Dabei ist nun der Inhalt der Folgebyteadresse das Lowerbyte, und der Inhalt der nachfolgenden Adresse (Überlauf dazu addiert) das Higherbyte.

Tritt ein Überlauf zwischen Überlauf und Inhalt der nächstfolgenden Adresse auf, so wird der Überlauf nicht berücksichtigt. Der Inhalt der aus Lower- und Higherbyte bestimmten Adresse wird mit dem Akkumulatorinhalt log. UND verknüpft. Das Ergebnis wird in den Akkumulator geladen.

Ohne Überlauf:

z.B. 31 07 AND (07),Y

Y-Registerinhalt ist z. B. 05. Inhalt der Speicherzelle 07 ist z. B. 01; 01 + 05 → 06. Inhalt des Akku's soll z. B. 09 sein. Der Inhalt der Speicherzelle 0006 sei z. B. 07. Nach der Ausführung des Befehls steht im Akku als Inhalt 01.

Mit Überlauf:

z.B. 31 07 AND (07),Y

Y-Registerinhalt ist z. B. 05. Inhalt der Speicherzelle 07 ist z.B. FE. FE + 05 → 1 (Überlauf) 03, d. h. Überlauf hat stattgefunden (Carry-Flag ist gesetzt). 03 ist nun das Lowerbyte und die Addition des Zelleninhaltes der Folgebyteadresse und dem Überlauf ist das Higherbyte. Z. B. Inhalt der Adresse 0008 ist 04. 04 und Überlauf (1) ist nun das Higherbyte. Lower- und Higherbyte ergeben nun die Adresse 0503. Der Inhalt dieser Adresse wird mit dem Akkumulatorinhalt logisch UND verknüpft.

Bemerkung:
Ist z. B. der Inhalt der Adresse 0008 = FF so ergibt die Addition mit dem Überlauf wieder einen Überlauf. Doch der neuerlich erzeugte Überlauf wird nicht berücksichtigt. Also FF + 1 → 100. Der Inhalt der Zelle 0003 wurde logisch UND verknüpft.

35 XX

Es wird das Folgebyte mit dem Inhalt des X-Registers addiert. Die Addition ist eine neue Adresse, deren Inhalt mit dem Akkumulatorinhalt logisch UND verknüpft wird.

z.B. 35 09 AND 09,X

X-Registerinhalt ist z. B. 04; Akkuinhalt sei 01. 09 + 04 → 0D Inhalt der Adresse 0D sei z. B. 0002. Nach der Ausführung wird im Akkumulator 00 stehen.

3D XX XX

Es wird die Adresse, die sich aus den beiden Folgebytes ergibt (Lower- und Higherbyte) mit dem Inhalt des X-Register's addiert. Dieses Additionsergebnis ergibt eine neue Adresse, deren Inhalt mit dem Akkumulatorinhalt logisch UND verknüpft wird. Das Ergebnis steht im Akku.

z.B. 35 00 03 AND $300,X

X-Registerinhalt ist z. B. 03; 0300 + 03 → 0303. Inhalt der Adresse 0303 sei z. B. 01. Nach der Ausführung wird 00 im Akkumulator stehen.

39 XX XX

Analoger Befehl zu 3D XX XX. Der Unterschied liegt nur in der Verwendung des Y-Register's.

z.B. 39 00 03 AND $300,Y

EOR

49 XX

Exklusives Oder

A	*B*	*T*
0	*0*	*0*
1	*0*	*1*
0	*1*	*1*
1	*1*	*0*

Es wird das Folgebyte mit dem momentanen Akkumulatorinhalt logisch EOR verknüpft. Das Ergebnis steht im Akku.

z.B. 49 31 EOR #$31

Momentaner Akkuinhalt sei z.B. 05
31 = 0 0 0 1 1 1 1 1
05 = 0 0 0 0 1 0 0 1
Nach der Befehlsausführung wird 0 0 0 1 0 1 1 0 = (16) Hex im Akku stehen.

4D XX XX

Es wird der Inhalt der Speicherzelle, die sich aus den 2 Folgebytes ergibt (Lower- und Higherbyte) mit dem momentanen Akkuinhalt log. EOR verknüpft. Das Ergebnis steht dann im Akku, wobei voriger Inhalt überschrieben wird.

z.B. 4D 02 03 EOR $302

Der Inhalt der Speicherzelle 0302 sei z. B. FE. Der momentane Akkuinhalt sei z. B. FF. Nach der Befehlsausführung wird im Akku 01 stehen.

45 XX

Der Inhalt der Speicherzelle, die durch das Folgebyte angegeben ist,

wobei es sich um eine Adresse in der Zeropage handelt wird mit dem Akkumulator log. EOR verknüpft. Das Ergebnis wird im Akku geladen, wobei voriger Inhalt überschrieben wurde.

z.B. *45 04* *EOR 04*

Momentaner Akkuinhalt sei z. B. 03. Inhalt der Zelle 00 04 sei 02. Nach Befehlsausführung wird 06 im Akku stehen.

41 XX

Es wird der Inhalt des X-Register's zum Folgebytewert addiert. Das Additionsergebnis ist eine Adresse in der Zeropage. Deren Inhalt ist das Lowerbyte, wobei der nächstfolgende Adresseninhalt das Higherbyte ist. Der Inhalt aus der Adresse die sich aus Lower- und Higherbyte ergibt wird mit dem momentanen Akkumulatorinhalt log. EOR verknüpft. Das Ergebnis wird in den Akku geladen.

Anmerkung:
Ein eventuell vorkommender Überlauf bei der Addition X-Register und Folgebyte wird nicht berücksichtigt !

z.B. *41 32* *EOR ($32,X)*

X-Registerinhalt sei 01; momentaner Akkuinhalt sei z. B. 07
01 + 32 → 33
Inhalt der Adresse 0033 sei z. B. 01
Inhalt der Adresse 0034 sei z. B. 02
Der Inhalt der daraus resultierenden Adresse 02 01 sei z. B. 05.
Nach der Befehlsausführung wird im Akku 02 stehen.

51 XX

Es wird der Inhalt der Adresse das das Folgebyte angibt zum Y-Register addiert. Das Additonsergebnis ist eine Adresse, deren Inhalt mit dem Akkuinhalt logisch EOR verknüpft wird. Das Ergebnis steht im Akku. Tritt aber bei der Addition ein Überlauf auf, so wird der Überlauf zu der nächstfolgenden Adresse die im Folgebyte angegeben ist, addiert. Dabei ist nun der Inhalt der Folgebyteadresse das Lowerbyte und der Inhalt der nachfolgenden Adresse und Überlauf das Higherbyte.
Der Inhalt der Adresse die sich aus Lower- und Higherbyte ergibt, wird mit dem momentanen Akkuinhalt log. EOR verknüpft. Das Ergebnis

steht im Akku.

Ohne Überlauf:

z.B. 51 02 EOR (02),Y

Inhalt der Speicherzelle 00 02 sei z.B. 01. Y-Registerinhalt sei z. B. 04; momentaner Akkuinhalt sei z. B. 08. 01 + 04 → 05 Der Inhalt der Speicherzelle 05 sei z. B. 06. Nach der EOR Verknüpfung wird 0E im Akku stehen.

Mit Überlauf:

z.B. 5102 EOR (02),Y

Inhalt der Speicherzelle 00 02 sei z.B. FF. Y-Registerinhalt sei z. B. 03; FF + 03 → 1 (Überlauf) 02 ist nun das Lowerbyte und die Addition des Zelleninhaltes Adresse nach der Folgebyteausgabe und Überlauf ist das Higherbyte; z. B. Inhalt der Adresse 03 ist z. B. 04 und Überlauf 1 dazuaddiert ergibt das Higherbyte 05. Lower- und Higerbyte ergeben die Adresse 05 02. Der Inhalt dieser Adresse wird mit dem Akkuinhalt log. EOR verknüpft.

Bemerkung:
Ist z. B. der Inhalt der Adresse 03 FF, so ergibt deren Addition mit dem Überlauf wieder einen Überlauf. Doch dieser neuerlich erzeugte Überlauf wird nicht berücksichtigt. Also FF + 1 → 1 00. Dann würde der Inhalt der Adresse 00 02 log. EOR verknüpft.

55 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Die Addition ist eine neue Adresse, deren Inhalt mit dem momentanen Akkuinhalt log. EOR verknüpft wird. Ergebnis steht im Akku.

z.B. 55 09 EOR 0,9X

X-Registerinhalt sei z. B. 02; momentaner Akkuinhalt sei z. B. 01; 09 + 02 → 0B
Inhalt der Adresse 0B sei z. B. 10. Nach der Befehlsausführung wird im Akku (11) Hex stehen.

5D XX XX

Es wird die Adresse, die sich aus den beiden Folgebytes ergibt (Lower-

und Higherbyte) mit den Inhalt des X-Register's addiert.
Dieses Additionsergebnis ergibt eine neue Adresse, deren Inhalt mit dem momentanen Akkuinhalt log. EOR verknüpft wird. Das Ergebnis steht im Akku.

z.B.   5D 00 01   EOR $100,X

X-Registerinhalt sei 01; 01 00 + 01 → 01 01. Inhalt der Adresse 01 01 sei z. B. F0; momentaner Akkuinhalt sei z. B. 0F. Nach der Befehlsausführung wird FF im Akkumulator geladen.

59 XX XX

Analoger Befehl zu 5D XX XX. Der Unterschied liegt in der Verwendung des Y-Register's.

z.B.   59 00 01   EOR $100,Y



09 XX

ODER FUNKTION

A	B	T
0	0	0
0	1	1
1	0	1
1	1	1

Es wird das Folgebyte mit dem momentanen Akkumulatorinhalt logisch durch Oder verknüpft. Ergebnis wird im Akku gespeichert.

z.B.   09 FE   ORA #$FE

Momantaner Akkuinhalt sei z. B. 01. Nach der Befehlsausführung wird FF der Inhalt des Akkumulator's sein.

0D XX XX

Es wird der Inhalt der Speicherzelle, die sich aus den beiden Folgebytes ergibt (Lower- und Higherbyte) mit dem momentanen Akkumulatorinhalt logisch ODER verknüpft.

z.B. 0D 00 01 ORA $100

Momentaner Akkuinhalt sei z. B. 0A. Inhalt der Adresse 01 00 sei z. B. 0B. Nach der Befehlsausführung wird 0B der Akkumulatorinhalt sein.

05 XX

Es wird der Inhalt der Speicherzelle die sich aus dem Folgebyte ergibt mit dem momentanen Akkumulatorinhalt logisch ODER verknüpft. Das Ergebnis wird in Akku geladen.

z.B. 05 03 ORA 03

Momentaner Akkuinhalt sei z. B. 08. Inhalt der Adresse 00 03 sei z. B. 09. Nach der Befehlsausführung wird der Akkuinhalt 09 sein.

01 XX

Es wird der Inhalt des X-Register's zum Folgebytewert addiert. Das Additionsergebnis ist eine Adresse in der Zeropage. Deren Inhalt ist das Lowerbyte, wobei der nächstfolgende Adresseninhalt das Higherbyte ist. Der Inhalt der aus Lower- und Higherbyte ergebenden Adresse wird mit dem Akkumulatorinhalt logisch ODER verknüpft. Ein evtl. Überlauf bei der Addition wird nicht berücksichtigt.

z.B. 01 05 ORA (05,X)

X-Registerinhalt sei z. B. 08; 05 + 08 → 0D. Inhalt der Adresse 000D sei z. B. 03. Inhalt der Adresse 00 0E sei z. B. 02. Momentaner Akkuinhalt sei z. B. 01. Inhalt der Adresse 02 03 sei z. B. 02. Nach der Befehlsausführung wird im Akkumulator 02 stehen.

11 XX

Es wird der Inhalt der Adresse des Folgebytes zum Y-Register addiert.

Das Additionsergebnis ist eine Adresse, deren Inhalt mit dem Akkumulatorinhalt logisch ODER verknüpft wird. Das Ergebnis steht anschließend im Akku.

Tritt bei der Addition ein Überlauf auf, d. h. Carry-Flag wird gesetzt, so wird dieser Überlauf zu der nächstfolgenden Adresse die im Folgebyte angegeben ist, addiert.

Dabei ist der Wert ohne Überlauf des Lowerbyte und die Addition Überlauf und Inhalt der nachfolgenden Adresse das Higherbyte. Tritt hier ein Überlauf auf, so wird er nicht berücksichtigt. Der Inhalt der Adresse, die sich aus Lower- und Higherbyte ergibt, wird mit dem Akkumulatorinhalt logisch ODER verknüpft.

Das Ergebnis wird im Akku gespeichert.

Ohne Überlauf:

z.B.   11 07   ORA ($11),Y

Y-Registerinhalt ist z. B. 04. Inhalt der Adresse 0007 ist z. B. 02 02 + 04 → 06. Der Inhalt der Speicherzelle 06 wird mit dem momentanen Akkumulatorinhalt log. ODER verknüpft. Ergebnis steht im Akkumulator.

Mit Überlauf:

z.B.   11 07   ORA ($11),Y

Y-Registerinhalt ist z. B. 04. Inhalt der Adresse 0007 sei z. B. FC; FC + 04 → 1 (Überlauf) 01.
Der Wert ohne Überlauf also 01 ist jetzt das Lowerbyte. Das Higherbyte ergibt sich aus der Addition des Überlaufes mit dem Inhalt der nächstfolgenden Folgebyteadresse. Zu unserem Fall Inhalt der Adresse 0008 + Überlauf. Es soll z. B. Inhalt der Adresse 0008 03 sein. Das Higherbyte ist dann 03 + 1 → 04. Der Inhalt von der Lower- und Higherbyte ergebenden Adresse wird mit dem momentanen Inhalt des Akkumulator logisch ODER verknüpft und im Akku geladen.

Anmerkung:
Ist z. B. der Inhalt der Adresse 0008 = FF wird die Addition mit dem Überlauf wieder einen Überlauf erzeugen. Es wird aber der neuerliche Überlauf nicht berücksichtigt bzw. weg gelassen, so daß FF + 1 00 ergibt.

15 XX

Es wird das Folgebyte mit dem X-Registerinhalt addiert. Die Additon ergibt eine neue Adresse, deren Inhalt mit dem momentanen Akku-inhalt logisch ODER verknüpft wird. Das Ergebnis der Verknüpfung steht dann im Akku.

z.B. 15 07 ORA 07,X

X-Registerinhalt sei z. B. 01. Momentaner Akkumulatorinhalt sei z. B. 0F. 07 + 01 → 08. Inhalt der Adresse 0008 sei z. B. FF. Nach der Befehlsausführung wird im Akkumulator FF stehen.

1D XX XX

Es wird die Adresse, die sich aus Lower- und Higherbyte ergibt mit dem Inhalt des X-Register's addiert. Die Addition ergibt wiederum eine neue Adresse, wobei deren Inhalt mit dem momentanen Akkuinhalt log. ODER verknüpft wird. Das Ergebnis steht wiederum im Akkumulator.

z.B. 1D 00 17 ORA $1700,X

X-Registerinhalt ist z. B. 01. Momentaner Akkuinhalt sei z. B. 02; 17 00 + 01 → 17 01. Inhalt von 17 01 sei z. B. FD. Nach der Ausführung des Befehls wird im Akkuinhalt FF stehen.

19 XX XX

Dieser Befehl entspricht dem 1D XX XX Befehl mit dem einen Unterschied, daß hier das Y-Register verwendet wird.

z.B. 19 00 17 ORA $1700,Y

SBC

E9 XX

A – M – C → A
Es wird das Folgebyte vom momentanen Akkumulatorinhalt subtrahiert. Gleichzeitig wird von diesem Ergebnis noch der invertierte Wert des Carry-Flags subtrahiert. Das Ergebnis wird im Akkumulator abgespeichert.

z.B. E9 02 SBC #$02
Momentaner Akkuinhalt sei FE; FE – 02 → FC. Nehmen wir an, daß das Carry-Flag vor der Befehlsausführung mit 0 gesetzt ist. Der invertierte Wert ergibt aber 1. FC – 1 → FB. Der Akkuinhalt nach der Befehlsausführung ist FB.

ED XX XX

Es wird der Inhalt der Adresse, die aus den beiden Folgebytes hervor geht vom momentanen Akkumulatorinhalt subtrahiert, gleichzeitig wird zu diesem Ergebnis der invertierte Wert des Carry-Flag subtrahiert.

z.B. ED 00 01 SBC $100

Momentaner Akkuinhalt sei z. B. 0F. Inhalt der Adresse 01 00 soll 03 sein. Carry-Flag 1 gesetzt d. h. 0F – 03 – 0 → 0C wird im Akku abgespeichert.

E5 XX

Es wird der Inhalt der Adresse, die aus dem Folgebyte hervorgeht vom momentanen Akkumulatorinhalt subtrahiert. Der invertierte Carry-Flag-Wert wird anschließend vom Ergebnis noch subtrahiert.

z.B. E5 03 SBC 03

Momentaner Akkuinhalt sei z. B. 04. Inhalt der Adresse 0003 sei 01. Carry-Flag nicht gesetzt: d. h. 04 – 01 – 01 → 02; 02 wird im Akku geladen.

E1 XX

Es wird der Inhalt des X-Register's zum Folgebyte addiert. Das Ergebnis dieser Addition ist eine Adresse in der Zeropage. Deren Inhalt ist das Lowerbyte und die nächstfolgende Adresse ist das Higherbyte. Der Inhalt der sich aus Lower- und Higherbyte ergebenden Adresse wird vom Akkumulatorinhalt subtrahiert.
Das invertierte Carry-Flag wird vom Ergebnis auch noch subtrahiert.

z.B. E1 03 SBC (03,X)

X-Registerinhalt ist z. B. 04; 03 + 04 → 07; Inhalt der Adresse 0007 ist z. B. BB. Inhalt der nächstfolgenden Adresse 0008 ist z. B. 01. Inhalt der somit errechneten Adresse 01 BB ist z. B. 01. Momentaner Akkuinhalt sei z. B. 04. Carry-Flag gesetzt: 04 – 01 – 0 → 03. 03 wird in Akku gespeichert.

F1 XX

Es wird der Inhalt der Adresse des Folgebytes zum Y-Registerinhalt addiert. Additionsergebnis ist eine neue Adresse, deren Inhalt mit dem investierten Carry-Flag vom momentanen Akkuinhalt subtrahiert wird. Tritt bei der Addition zu Folgebyte und Y-Register ein Überlauf auf, so wird der Überlauf zu der nächstfolgenden Adresse die im Folgebyte angegeben ist addiert. Dabei ist der Wert ohne Überlauf das Lowerbyte und die Addition Überlauf und Inhalt der nächstfolgenden Adresse das Higherbyte. Tritt hier ein Überlauf auf, so wird er nicht berücksichtigt. Der Inhalt der Adresse die sich aus Lower- und Higherbyte ergibt und das invertierte Carry-Flag wird vom momentanen Akkuinhalt subtrahiert.

Ohne Überlauf:

z.B. F1 07 SBC (07),Y

Y-Registerinhalt sei z. B. 04. Inhalt der Speicherzelle 0007 sei z. B. 01; 01 + 04 → 05. Der Inhalt der Speicherzelle 0005 sei z. B. 05. Momentaner Akkuinhalt z. B. 0F.
Carry-Flag nicht gesetzt. Nach Befehlsausführung wird im Akku 0F – 05 – 01 → 09 stehen. Carry-Flag ist gesetzt.

Mit Überlauf:

z.B. F1 07 SBC (07),Y

Y-Registerinhalt sei z. B. 04. Inhalt der Speicherzelle 0007 sei z. B. FE; FE + 04 → 1 (Überlauf) 02. 02 ist Lowerbyte Folgebyte + 1 also 0008 sei z. B. 01 + Überlauf 1 ergibt 02; 02 ist Higherbyte. Lower- und Higherbyte ergibt Adresse 0202, Inhalt z. B. 05; momentaner Akkuinhalt 0F. Carry-Flag gesetzt.
Nach Befehlsausführung wird 0F – 05 – 00 = 0A im Akku stehen. Carry-Flag ist gesetzt.

F5 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Die Addition ist eine neue Adresse, deren Inhalt mit dem invertierten Carry-Flag vom momentanen Akkuinhalt subtrahiert wird.

z.B.   F5 07   SBC $07,X

X-Registerinhalt sei z. B. 01; 07 + 01 → 08. Inhalt der Adresse 08 sei z. B. FE; momentaner Akkuinhalt sei z. B. 05. Carry-Flag nicht gesetzt. Nach Befehlsausführung wird 05 – FE – 01 = also 06 (–250) Dez im Akku stehen. Carry-Flag nicht gesetzt.

FD XX XX

Es wird die Adresse, die sich aus den 2 Folgebytes ergibt (Lower- und Higherbyte) und der Inhalt des X-Register's addiert. Dieses Additionsergebnis ergibt eine neue Adresse, deren Inhalt mit dem invertierten Carry-Flag vom momentanen Akkuinhalt subtrahiert wird.

z.B.   FD 00 01   SBC $100,X

X-Registerinhalt sei z. B. 02; 0100 + 02 → 0102. Inhalt von 0102 sei z. B. 02; Akkuinhalt sei 0E, Carry-Flag gesetzt. Nach der Befehlsausführung wird im Akkuinhalt 0E – 02 – 0 → 0C stehen. Carry-Flag bleibt gesetzt.

F9 XX XX

Dieser Befehl ist der analoge Befehl zum FD XX XX Befehl, nur daß hier das Y-Register angesprochen wird.

z.B.   F9 00 01   SBC $100,Y

DEC

CE XX XX

Es wird der Inhalt der Speicherzelle, die sich aus den 2 Folgebytes ergibt (Lower- und Higherbyte) um 1 erniedrigt und in derselben Adresse wieder abgelegt.

z.B.   CE 00 01   DEC $100

Der Inhalt von Adresse 0100 soll z. B. 05 sein. Von diesen 5 wird 1 subtrahiert und in Speicherzelle 0100 gespeichert.

C6 XX

Es wird der Inhalt der Speicherzelle des Folgebytes (Zeropage) um 1 erniedrigt und in derselben Adresse wieder abgelegt.

z.B.   C6 01   DEC 01

Der Inhalt der Adresse 0001 wird um 1 erniedrigt und in derselben Speicherzelle abgelegt, also Inhalt von 0001 vorher sei 07. Inhalt von 0001 nachher ist 06.

D6 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Die Addition ergibt eine neue Adresse, deren Inhalt um 1 erniedrigt wird. Tritt ein Überlauf bei der Addition auf, so wird der Überlauf nicht berücksichtigt.

z.B.   D6 08   DEC 08,X

X-Registerinhalt sei 01; 08 + 01 → 09. Inhalt der Adresse 09 sei z. B. FF, dieser Inhalt wird um 1 erniedrigt, so daß danach in Speicherzelle 0009 der Inhalt FE steht.

DE XX XX

Es wird die Adresse, die sich aus den 2 Folgebytes ergibt (Lower- und Higherbyte), und der Inhalt des X-Register's addiert. Die Additon ergibt eine neue Adresse, deren Inhalt um 1 erniedrigt wird.

z.B. *DE 00 11* *DEC $1100,X*

X-Registerinhalt sei 01; 1100 + 01 → 1101. Inhalt der Adresse 1101 z. B. 08 wird um 1 erniedrigt, so daß danach 07 darin steht.



CA

Es wird der Inhalt des X-Register's um 1 erniedrigt und in X-Register wieder abgespeichert.

z.B. *X-Registerinhalt 05. Nach Ausführung des Befehls steht 04 im X-Register.*



88

Es ist der analoge Befehl zum DEX-Befehl nur, daß hier das Y-Register verwendet wird.

INC

EE XX XX

Es wird der Inhalt der Speicherzelle die sich aus den 2 Folgebytes ergibt, (Lower- und Higherbyte) um 1 erhöht und wieder in der selben Adresse abgelegt.

z.B. EE 00 02 INC $200

Inhalt der Zelle 0200 soll 08 sein. Nach der Ausführung steht 09 in der Zelle 0200.

E6 XX

Es wird der Inhalt der Speicherzelle die das Folgebyte angibt, um 1 erhöht und in derselben Zelle abgelegt.

z.B. E6 D0 INC $D0

Inhalt von 00D0 sei 08. Nach der Ausführung steht in Adresse 00D0 09.

F6 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Das Ergebnis ist eine Adresse in der Zeropage. Deren Inhalt wird um 1 erhöht und in der gleichen Adresse abgelegt.
Ein eventuell vorkommender Überlauf bei der Addition wird nicht berücksichtigt.

z.B. F6 06 INC 06,X

X-Registerinhalt sei 02; 06 + 02 → 08. Inhalt der Speicherzelle 0008 sei 01. Nach Ausführung des Befehls steht 0B in Adresse 0008.

FE XX XX

Es wird die Adresse die sich aus den beiden Folgebytes ergibt, (Lower- und Higherbyte) und der Inhalt des X-Register's addiert. Das Ergebnis der Addition ist eine Adresse, deren Inhalt um 1 erhöht wird. Das Ergebnis steht wieder in derselben Speicherzelle.

z.B. FE 01 03 INC $301,X

X-Registerinhalt sei 05; 0301 + 05 → 0306. Der Inhalt von 0306 soll z. B. 0F sein. Nach Ausführung des Befehls steht in Speicherzelle 0301 10.



E8

Es wird der Inhalt des X-Register's um 1 erhöht und anschließend im X-Register wieder gespeichert.

z.B. Momentaner X-Registerinhalt 09. Nach Ausführung des Befehls steht 0A im X-Register.



C8

Der analoge Befehl zum INX-Befehl. Der Unterschied liegt nur in der Benutzung des Y-Register's.

ASL

OE XX XX

C ← [7 0] ← 0

Es wird der Inhalt der Speicherzelle die sich aus den Folgebytes ergibt, (Lower- und Higherbyte) um 1 Stelle nach links verschoben wobei von der rechten Seite eine 0 eingeschoben wird, und das herausgeschobene Bit im Carry-Flag gespeichert wird.

z.B. OE 00 03 ASL $300

Inhalt von 0300 sei z. B. (81) 16 = (1000 0001) nach der Ausführung des Befehls wird in der Speicherzelle 0300 die Zahl 2 stehen und das Carry-Flag ist mit 1 gesetzt.

Bildlich: *Zelle 300*

	1 0 0 0 0 0 0 1	*vorher*
Carry-Flag ← [1] ist 1	*0 0 0 0 0 0 1 0*	*nachher*

06 XX

Es wird der Inhalt der Speicherzelle die durch das Folgebyte angegeben ist, um 1 Stelle nach links verschoben unter gleichzeitiger Aufnahme von der rechten Seite einer 0.

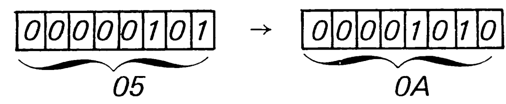
z.B. 06 04 ASL 04

Inhalt von 04 sei 1. Nach der Ausführung steht in Adresse 0004 eine 2 und Carry-Flag ist 0. Siehe auch OE XX XX.

0A *ASL (oder auch ASL A)*

Der Akkumulatorinhalt wird unter gleichzeitiger Einschiebung einer 0 von rechts nach links verschoben. Das Carry-Flag wird mit dem herausfallenden Bit gesetzt.

z.B. Akkuinhalt sei 05
Nach der Befehlsausführung steht 0A im Akku.



16 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Das Ergebnis ist eine neue Adresse, deren Inhalt unter Einschiebung von rechts einer 0 um 1 Stelle nach links verschoben wird. Das herausfallende Bit wird im Carry-Flag gesetzt.

z.B. 16 05 ASL 05,X

X-Registerinhalt sei 08; 05 + 08 → 0D; angenommener Inhalt von 000B sei 80. Nach der Ausführung ist in 000D die Zahl 0 gespeichert. Das Carry-Flag ist mit 1 gesetzt worden.

[1|0|0|0|0|0|0|0] [0|0|0|0|0|0|0|0]

Ein eventueller Überlauf bei der Addition wird nicht berücksichtigt !

1E XX XX

Es wird die Adresse die sich aus den 2 Folgebytes ergibt (Lower- und Higherbyte) und der Inhalt des X-Register's addiert. Das Additionsergebnis ist eine Adresse, deren Inhalt nach links verschoben wird. Rechts wird eine 0 eingeschoben und der linke Überlauf wird im Carry-Flag gesetzt.

z.B. 1E 00 01 ASL $100,X

X-Registerinhalt sei z. B. 05; 0100 + 05 → 0105. Der Inhalt der Adresse 0105 sei z. B. 08. Nach der Befehlsausführung steht in der Adresse 105 die 10.

Carry-Flag 0 ← [0|0|0|0|1|0|0|0] ← 0 [0|0|0|1|0|0|0|0]
8 16

LSR

4E XX XX

0 → [7 0] → C

Es wird der Inhalt der Speicherzelle die sich aus den Folgebytes ergibt, (Lower- und Higherbyte) um 1 Stelle nach rechts verschoben, wobei von der linken Seite eine 0 eingeschoben wird, und der Überlauf der Verschiebung im Carry-Flag gesetzt ist.

z.B. 4E 00 02 LSR $200

Inhalt von 0200 sei z. B. (03) 16 = (0000 0011) nach der Ausführung des Befehls wird in Adresse 0200 eine 1 der Inhalt sein. Die 1, die hinausgeschoben wurde, steht nun im Carry-Flag.

C

[0|0|0|0|0|0|0|1] – [1]

46 XX

Es wird der Inhalt der Speicherzelle die durch das Folgebyte angegeben ist, um 1 nach rechts verschoben unter gleichzeitiger Aufnahme einer 0 von der linken Seite.

z.B. 06 01 LSR 01

Inhalt momentan sei die Hex Zahl 0F. Nach der Ausführung des Befehls wird 07 der Inhalt von Adresse 0001 sein, und das Carry-Flag wird 1 gesetzt sein.

4A *LSR (oder aus LSR A)*

Der Akkumulatorinhalt wird unter gleichzeitiger Einschiebung einer 0 von links nach rechts verschoben. Das somit frei werdende Bit von der rechten Seite wird im Carry-Flag gesetzt.

z.B. *Inhalt des Akku sei z. B. FE*
Nach der Ausführung des Befehls wird im Akku 7F stehen und das Carry-Flag ist mit 0 gesetzt.

| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | → | 0 | Carry-Flag

56 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Das Additionsergebnis ist eine neue Adresse, deren Inhalt um 1 nach rechts verschoben wird unter gleichzeitiger Aufnahme eine 0 von der linken Seite. Das Carry-Flag wird mit dem freiwerdenden Bit von der rechten Seite gesetzt.

Bemerkung:
Ein eventueller Überlauf bei der Addition wird nicht berücksichtigt.

z.B. *56 FE* *LSR $FE,X*

X-Registerinhalt 0F; FE + 0F → 10D. Da der Überlauf nicht berücksichtigt wird, wird der Inhalt der Adresse 000D um 1 Stelle nach rechts verschoben.

5E XX XX

Es wird die Adresse, die sich aus den zwei Folgebytes ergibt (Lower- und Higherbyte), und der Inhalt des X-Register's addiert. Das Additonsergebnis ist eine Adresse, deren Inhalt nach rechts um eine Stelle verschoben wird. Das rechte Bit das dabei frei wird, wird im Carry-Flag gesetzt.

z.B. *5E 00 01* *LSR $100,X*

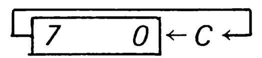
X-Registerinhalt sei z. B. 08. Der Inhalt der Adresse 0108 sei z. B. 07. Nach der Befehlsausführung steht in der Adresse 108 der Inhalt 03; das Carry-Flag ist mit 1 gesetzt.

0 → | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | → | 1 | Carry-Flag

ROL

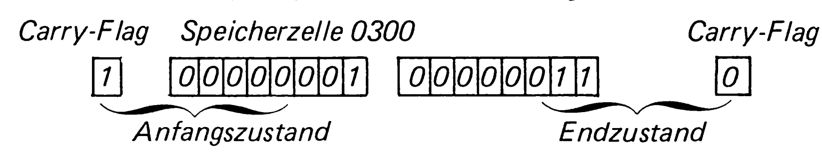
2E XX XX



Es wird der Inhalt der Speicherzelle, die sich aus den Folgebytes ergibt (Lower- und Higherbyte) rolliert, d. h. das linke Bit wird in das Carry-Flag gebracht, wobei der vorige Carry-Flag-Inhalt rechts wieder eingeschoben wird.

z.B. 2E 00 03 ROL $300

Inhalt von 0300 sei 01, das Carry-Flag sei 1. Nach der Ausführung des Befehls wird in der Speicherzelle 0300 der Inhalt 03 sein. Das Carry-Flag wird anschließend 0 gesetzt sein.



26 XX

Es wird der Inhalt der Speicherzelle die durch das Folgebyte angegeben ist, folgender Änderungen unterzogen, unter zu Hilfe nahme des Carry-Flags wird Speicherzelleninhalt nach links verschoben in das Carry-Flag hinein, der vorige Carry-Flag-Inhalt wird von rechts in den Speicherzelleninhalt geschoben.

z.B. 26 0F ROL $0F

Inhalt von 0F sei z. B. 02. Das Carry-Flag sei z. B. 0. Nach der Ausführung wird der Inhalt von 0F 04 sein. Das Carry-Flag wird wieder mit 0 geladen.

2A *ROL (oder auch ROL A)*

Der Akkumulatorinhalt wird um eine Stelle nach links verschoben. Der momentane Inhalt des Carry-Flags wird von rechts eingeschoben; das

Bit das hinausgeschoben wurde wird im Carry-Flag neu gesetzt.

z.B. Akkuinhalt FF, Carry-Flaginhalt sei 0. Nach der Befehlsausführung steht FE als Inhalt im Akku. Das Carry-Flag ist 1 gesetzt.

36 XX

Es wird das Folgebyte mit dem Inhalt des X-Registers addiert. Das Ergebnis ist eine neue Adresse, die um eine Stelle nach links verschoben wird. Der momentane Carry-Flag-Inhalt wird von rechts eingeschoben. Das Bit das aus der Speicherzelle hinausgeschoben wurde, wird als neues Bit im Carry-Flag gesetzt.

Bemerkung:
Entsteht bei der Addition ein Überlauf, so wird er nicht berücksichtigt.

z.B. 36 03 ROL 03,X

X-Registerinhalt F0 Carry-Flag ist 1; 03 + F0 → F3. Der Inhalt der Adresse F3 sei z. B. FF. Nach der Befehlsausführung wird wieder FF der Inhalt der Adresse F3 sein.

3E XX XX

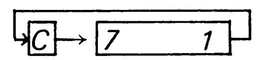
Es wird die Adresse, die sich aus den 2 Folgebytes ergibt (Lower- und Higherbyte) und der Inhalt des X-Register's addiert. Das Additionsergebnis ist eine neue Adresse, deren Inhalt um 1 Stelle nach links verschoben wird. Der momentane Inhalt des Carry-Flags wird von rechts eingeschoben. Das Bit, das aus der Speicherzelle hinausgeschoben wurde, wird im Carry-Flag neu gesetzt.

z.B. 3E 00 01 ROL $100,X

X-Registerinhalt sei z. B. 05; 0100 + 05 → 0105. Der Inhalt der Adresse 0105 sei z. B. 04, Carry-Flag sei 0. Nach der Befehlsausführung steht in der Adresse 105 08 und das Carry-Flag wird wieder 0 gesetzt.

ROR

6E XX XX



Es wird der Inhalt der Speicherzelle, die sich aus den Folgebytes ergibt, (Lower- und Higherbyte) um eine Stelle nach rechts verschoben. Der momentane Inhalt des Carry-Flags wird von links eingeschoben und das Bit das hinausgeschoben wurde, ist der neue Inhalt des Carry-Flags.

z.B. 6E 00 01 ROR $100

Inhalt der Speicherzelle 100 sei z. B. 02, Carry-Flag-Inhalt sei 1. Nach der Befehlsausführung steht 81 im Inhalt der Speicherzelle 100. Das Carry-Flag ist anschließend mit 0 gesetzt.

66 XX

Es wird der Inhalt der durch das Folgebyte angegebenen Speicherzelle in der Zeropage um 1 Stelle nach rechts geschoben. Der momentane Inhalt des Carry-Flags wird von links eingeschoben und das Bit, das aus der Speicherzelle rechts hinausgeschoben wurde, ist der neue Inhalt des Carry-Flags.

z.B. 66 E0 ROR $E0

Inhalt von E0 sei z. B. 04, Carry-Flag sei z. B. 0. Nach der Ausführung des Befehls wird der Inhalt von Speicherzelle E0 02 sein. Das Carry-Flag wird nach wie vor 0 sein.

6A *ROR (oder auch ROR A)*

Der Akkumulatorinhalt wird um eine Stelle nach rechts verschoben. Der momentane Inhalt des Carry-Flags wird von links eingeschoben. Das Bit das hinausgeschoben wurde, ist der neue Inhalt des Carry-Flags.

76 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Entsteht bei dieser Addition ein Überlauf, so wird er nicht berücksichtigt. Das Additionsergebnis wird um 1 Stelle nach rechts verschoben. Der momentane Carry-Flag-Inhalt wird von links in die Speicherzelle eingeschoben. Das Bit das aus der Speicherzelle hinausgeschoben wurde ist neuer Inhalt des Carry-Flags.

z.B. 76 0F ROR $0F,X

X-Registerinhalt sei 01, Carry-Flag soll 1 gesetzt sein.
0F + 01 → 10. Der Inhalt von 0010 soll z. B. FE sein. Nach der Befehlsausführung wird FF in der Speicherzelle stehen und 0 wird im Carry-Flag sein.

7E XX XX

Es wird die Adresse, die sich aus den zwei Folgebytes ergibt (Lower- und Higherbyte) und der Inhalt des X-Register's addiert. Das Additionsergebnis wird um 1 Stelle nach rechts verschoben. Der momentane Carry-Flag-Inhalt wird von links in die Speicherzelle eingeschoben. Das Bit das bei der Verschiebung der Speicherzelle hinausgeschoben wurde, ist nun neuer Inhalt des Carry-Flags.

z.B. 7E 00 02 ROR $200,4

X-Registerinhalt sei z. B. 01. 0200 + 01 → 0201. Der Inhalt der Speicherzelle 0201 sei z. B. 10. Das Carry-Flag soll 0 gesetzt sein. Der Inhalt von 0201 ist nach der Befehlsausführung 08. Das Carry-Flag bleibt auf 0 gesetzt.

CMP

A—M

Allgemeines:
Bei diesem Befehl wird nur das Statusregister geändert. Alle anderen Registerinhalte bleiben erhalten. Der CPM-Befehl wird nur im Zusammenhang mit Verzweigungsbefehlen verwendet.

BCC ermittelt	*A ⟨ M*
BEQ ermittelt	*A = M*
BCS folgend auf BEQ	*A ⟩ M*

C9 XX

Es wird das Folgebyte vom momentanen Akkumulatorinhalt subtrahiert. Nach dem erläuterten Schema werden die 3 Flags des Statusregister's gesetzt.

z.B. C9 01 CPM #$01

CD XX XX

Es wird der Inhalt der Speicherzelle, die sich aus den Folgebytes ergibt vom momentanen Akkuinhalt subtrahiert. Nach dem erläuterten Schema werden die 3 Flags des Statusregister's gesetzt.

z.B. CD 00 01 CMP $100
CMP $01

C5 XX

Es wird der Inhalt der Speicherzelle die im Folgebyte angegeben ist vom momentanen Akkuinhalt subtrahiert und das Ergebnis wird nach unten erläuterten Schema benutzt, die 3 Statusregisterflags zu setzen.

z.B. C5 01 CMP 01

Schema der Flagsetzung

	N	C	Z
Akku ⟨ Mem.	*SET* *Für Vergleich im 2-er Komplement*	*RESET*	*RESET*
Akku = Mem.	*RESET*	*SET*	*SET*
Akku ⟩ Mem.	*RESET* *Für Vergleich im 2-er Komplement*	*SET*	*RESET*

Beispiel:

A = FE, M = 00
absolut Akku ⟩ Mem   *C = 1, Z = 0*
im 2-er Komplement
Akku ⟨ Mem   *N = 1*

A = 00, M = FE
absolut Akku ⟨ Mem   *C = 0, Z = 0*
im 2-er Komplement
Akku ⟩ Mem   *N = 0*

C1 XX

Es wird der Inhalt des X-Register's zum Folgebyte addiert. Das Ergebnis dieser Addition ist eine Adresse in der Zeropage. Deren Inhalt ist das Lowerbyte und die nächstfolgende Adresse ist das Higherbyte. Der Inhalt der Adresse die sich aus Lower- und Higherbyte ergibt wird vom momentanen Akkumulatorinhalt subtrahiert. Das Ergebnis setzt nach der Tabelle die 3 Flags des Statusregisters.

z.B. C1 01 CMP (01,X)

D1 XX

Es wird der Inhalt der Adresse, die das Folgebyte ergibt zum Y-Registerinhalt addiert. Dies ergibt eine neue Adresse, deren Inhalt vom Akkumulator subtrahiert wird.

Tritt aber bei der Addition zu Folgebyteadresseninhalt und Y-Register ein Überlauf auf, so wird der Überlauf zu der nächstfolgenden Adresse die im Folgebyte angegeben ist addiert. Dabei ist der Wert ohne Überlauf das Lowerbyte und die Addition Überlauf und Inhalt der nächstfolgenden Adresse das Higherbyte. Tritt hier ein Überlauf auf, so wird er nicht berücksichtigt. Der Inhalt der Adresse die sich aus Lower- und Higherbyte ergibt wird vom Akkumulatorinhalt subtrahiert. Das Ergebnis legt die Flagsetzung des Statusregister's fest.

z.B.   D1 01   CMP (01),Y

D5 XX

Es wird das Folgebyte mit dem Inhalt des X-Register's addiert. Die Addition ist eine neue Adresse, deren Inhalt vom momentanen Akkuinhalt subtrahiert wird. Das Ergebnis setzt die Zustände der 3 Flags im Statusregister fest.

z.B.   D5 01   CMP 01,X

DD XX XX

Es wird die Adresse, die sich aus den 2 Folgebyte ergibt, Lower- und Higherbyte mit dem X-Registerinhalt addiert. Das Additionsergebnis ergibt eine neue Adresse, deren Inhalt vom momentanen Akkuinhalt subtrahiert wird. Dieses Ergebnis setzt die Zustände der 3 Flags des Statusregister's fest.

z.B.   DD 00 10   CMP $1000,X

D9 XX XX

Ist der analoge Befehl zu DD XX XX, nur im Unterschied dazu, daß das Y-Register verwendet wird.

z.B.   D9 00 10   CMP $1000,Y

CPX

X–M

Allgemeines:
Bei diesen Befehl wird kein Register oder Speicherinhalt geändert. Der CPX-Befehl wird nur im Zusammenhang von Verzweigungen verwendet.

BCC ermittelt	$X < M$
BEQ ermittelt	$X = M$
BCS folgend auf BEQ	$X > M$

E0 XX

Es wird das Folgebyte vom X-Registerinhalt subtrahiert. Nach dem erläuterten Schema werden die 3 Flags des Statusregisters gesetzt.

z.B. E0 01 CPX #01

EC XX XX

Es wird der Inhalt der Speicherzelle, die sich aus den Folgebytes ergibt, vom X-Registerinhalt subtrahiert. Nach dem erläuterten Schema werden die 3 Flags des Statusregister's gesetzt.

z.B. EC 00 10 CPX $100

E4 XX

Es wird der Inhalt der Speicherzelle die sich aus dem Folgebyte ergibt, vom momentanen X-Registerinhalt subtrahiert. Nach dem unten erläuterten Schema werden die Flags gesetzt.

z.B. E4 0A CPX $0A

Schema der Flagsetzung

	N	C	Z
Akku ‹ Mem.	*SET* *Für Vergleich im 2-er Komplement*	*RESET*	*RESET*
Akku = Mem.	*RESET*	*SET*	*SET*
Akku › Mem.	*RESET* *Für Vergleich im 2-er Komplement*	*SET*	*RESET*

z.B. *X-Register 04*
E0 05 → *Statusregister (B0) Hex = (176) Dez*
|1 |0 1 1 0 0 |0 |0 |

X-Register 05
E0 05 → *Statusregister (33) Hex = (51) Dez*
0 0 1 1 0 0 |1 |1

X-Register 06
E0 05 → *Statusregister (31) Hex = (49) Dez*
|0 |0 1 1 0 0 |0 |1



Allgemeines:

Bei diesem Befehl wird kein Adresseninhalt außer dem Statusregister geändert. Der CPM-Befehl wird nur im Zusammenhang mit Verzweigungsbefehlen verwendet.

BCC ermittelt $Y < M$
BEQ ermittelt $Y = M$
BEQ folgend auf BEQ $Y > M$

C0 XX *CPY #$01*

Es wird das Folgebyte vom Y-Register subtrahiert. Nach dem erläuterten Schema werden die 3 Flags des Statusregisters gesetzt.

CC XX XX *CPY $1000*

Es wird der Inhalt der Speicherzelle, die sich aus den Folgebytes ergibt, vom Y-Register subtrahiert. Nach dem erläuterten Schema werden die 3 Flags des Statusregister's gesetzt.

C4 XX *CPY $01*

Es wird der Inhalt der Speicherzelle, die sich aus den Folgebyte ergibt vom Y-Registerinhalt subtrahiert. Das Ergebnis wird zum setzen der 3 Flags benutzt.

Schema der Flagsetzung

	N	C	Z
Akku $<$ Mem.	SET Für Vergleich im 2-er Komplement	RESET	RESET
Akku = Mem.	RESET	SET	SET
Akku $>$ Mem.	RESET Für Vergleich im 2-er Komplement	SET	RESET

z.B. *Y-Register 04*

C0 05 → *Statusregister (B0) Hex*

1 0 1 1 0 0 0 0

Y-Register 05

C0 05 → *Statusregister (33) Hex*

0 0 1 1 0 0 1 1

Y-Register 06

C0 05 → *Statusregister (31) Hex*

0 0 1 1 0 0 0 1



2C XX XX

UND FUNKTION

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Der Inhalt der Speicherzelle die sich aus den beiden Folgebytes ergibt (Lower- und Higherbyte) wird mit dem Inhalt des Akkumulator's logisch UND verknüpft. Bit 6 und Bit 7 der Verknüpfung werden im Prozessorstatusregister gesetzt.

z.B.   2C 00 01   BIT $100

Angenommen der Inhalt der Speicherzelle 0100 sei 81. Der angenommene Inhalt des Akkus sei FF.

```
UND verknüpft:    1 0 0 0 0 0 0 1
                  1 1 1 1 1 1 1 1
                  ---------------
                  1 0 0 0 0 0 0 1
                 /   \
             Bit 7   Bit 6
```

Das Prozessorstatusregister zeigt nach der Operation folgenden Inhalt:

1 0 X X X X X X   / X bedeutet beliebig

bei unseren Test B0 = 1 0 1 1 0 0 0 0

24 XX

Der Inhalt der Speicherzelle die durch das Folgebyte angegeben ist wird mit Akkumulatorinhalt logisch UND verknüpft. Bit 6 und Bit 7 der Verknüpfung werden im Statusregister abgelegt.

z.B.   24 03   BIT 03

Angenommener Inhalt von 0003 sei C1 und Akkuinhalt sei FF.

```
UND verknüpft:      1 1 0 0 0 0 0 1 = C1
                    1 1 1 1 1 1 1 1 = FF
                    ----------------
                    1 1 0 0 0 0 0 1
                   /   \
                Bit 7   Bit 6
```

Statusregister hat nach der Operation folgenden Inhalt:
1 1 X X X X X X   bei unserem Test also 1 1 1 1 0 0 0 0

BCC

90 XX

Es wird das Carry-Flag auf 0 überprüft. Wenn die Bedingung erfüllt ist, wird zu der Adresse gesprungen die sich aus dem Folgebyte ergibt. Es können 127 Adressen vorwärts und 128 Adressen rückwärts gesprungen werden.

Bemerkung:

Carry-Flag ist Bit 0 des Statusregisters.

□□□□□□□■
76543210

z.B.   90 03   BCC 03

BCS

B0 XX

Es wird das Carry-Flag auf 1 überprüft. Ansonsten gilt das gleiche wie bei BCC.

z.B.   B0 $FC   BCS $FC

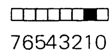
BEQ

F0 XX

Es wird das Zero-Flag auf 1 überprüft. Tritt die Bedingung zu, so wird zu der Adresse gesprungen, die sich aus dem Folgebyte ergibt. 127 Vorwärtssprünge und 128 Rückwärtssprünge sind möglich.

Bemerkung:

Zero-Flag ist Bit 1 des Status-Registers.



z.B. F0 F4 BEQ $F4

Falls das Zero-Flag 1 gesetzt ist, wird 4 Adressen rückwärts gesprungen.

BNE

D0 XX

Es wird das Zero-Flag auf 0 überprüft. Ansonsten der selbe Verlauf wie beim BEQ-Befehl.

z.B. D0 FE BNE $FE



30 XX

Es wird das Negativ-Flag auf 1 überprüft. Trifft die Bedingung zu, so wird zu der Adresse gesprungen, die sich aus den Folgebyte ergibt.

Bemerkung:

Das Negativ-Flag ist Bit 7 im Status-Register.

■□□□□□□□
76543210

z.B.   30 08   BMI 08

BPL

10 XX

Es wird das Negativ-Flag auf 0 überprüft. Ansonsten der selbe Befehl wie BMI.

z.B.   10 08   BPL 08

BVC

50 XX

Es wird das Overflow-Flag auf 0 überprüft. Trifft die Bedingung zu, so wird zu der Adresse gesprungen, die sich aus dem Folgebyte ergibt.

Bemerkung:

Overflow-Flag ist Bit 6 im Status-Register.

□■□□□□□□
76543210

z.B.   60 07   BVC 07

BVS

70 XX

Es wird das Overflow-Flag auf 1 überprüft. Ansonsten der selbe Befehl wie BVC.

z.B. 70 0F BVS $0F

Beispiel zu den Verzweigungsbefehlen: Berechnung der relativen Adresse. Es soll am BNE-Befehl erklärt werden. Trifft sonst für alle Verzweigungsbefehle zu.

Rückwärtssprung		*Ziel (Sprung)*
0200 E1	*LDX FF*	*AZ FF*
0202	*DEX*	*CA*
0203	*BNE E1*	*D0 FB*
0205	*END*	*00*

Vorwärtssprung		
0300	*LDX 03*	*A2 01*
0302 E2	*DEX*	*CA*
0303	*BEQ E3*	*F0 03*
0305	*JMP E2*	*4C 02 02*
0308 E3	*END*	*00*
		(Ziel) Sprung

Beide Programme machen das gleiche; sie zählen jeweils das X-Register des Anfangs mit FF geladen wurde auf 0 herunter. Zu der Sprungadressenbestimmung in den Verzweigungsbefehlen.

Vorwärtssprung:
Es werden die zu überspringenden Adressen ab dem Folgebyte in der Verzweigeanweisung. In unserem Fall sind es 3 Adressen die übersprungen werden. Also 03 ist das Folgebyte.

Rückwärtssprung:
Wie beim Vorwärtssprung wird die Anzahl der zu überspringenden Adressen vom Folgebyte des Verzweigungsbefehles aus gezählt. Da es sich aber um einen Rückwärtssprung handelt muß

man folgendermaßen vorgehen. Ziehen Sie die ermittelten Adressprunge, in unserem Beispiel 5 von der Binärzahl 11111111 ab und addieren Sie zu den Ergebnis eine 1 dazu. Vorsicht mit den Zahlensystemen !

In unserem Beispiel:

```
1 1 1 1 1 1 1 1
0 0 0 0 0 1 0 1
---------------
1 1 1 1 1 0 1 0
+             1
---------------
1 1 1 1 1 0 1 1   = FB
```

FB ist das Folgebyte.

BRK

00

Das Break-Flag wird automatisch im Statusregister gesetzt. Es wird unterschieden zwischen einem Programmbreak und einem Hardware-Interrupt. Keine anderen Benutzerinstruktionen werden modifiziert.



4C XX XX

Absoluter Sprung zu der Adresse, die im Operanden in Lower- und Higherbyte angegeben ist.

z.B. 4C 00 02 JMP $200

Es wird bei Adresse 0200 im Programm fortgefahren.

6C XX XX

Indirekter Sprung zu der Adresse, deren Lower- und Higherbyte sich folgendermaßen ergibt. Die 2 Folgebytes sind 1 Adressenangabe, deren Inhalt das Lowerbyte ist. Der nächstfolgende Adresseninhalt ist das Higherbyte.

z.B. 6C 01 03 JMP ($301)

Inhalt der Adresse 0301 sei z. B. 00
Inhalt der Adresse 0302 sei z. B. 02
somit erfolgt Sprung an Adresse 0200

JSR

20 XX XX

Es erfolgt ein absoluter Sprung in eine Unterroutine, die entweder selbst eingegeben wurde oder bereits fest besteht. Die Folgebytes sind Lower- und Higherbyte der Sprungadresse.

z.B. 20 19 1F JSR $1F19

Es erfolgt ein Unterroutinensprung zu Start-Adresse 1F 19.

RTS

60

Bei selbst geschriebenen Unterroutinen muß der letzte Befehl ein RTS-Befehl sein. Das bedeutet, daß im eigentlichen Programm fortgefahren wird.

RTI

40

Zurückspeichern des Statusregister und des Programmcounter welche beide im Stack gespeichert waren. Neufestlegung des Stackpointer. Eigentliche Bedeutung Rücksprung von einem Interrupt.

NOP

EA

Es wird keine Operation ausgeführt. Alle Registerinhalte bleiben erhalten. Es wird zur Zeitverzögerung oder zum Programmauffüllen verwendet.

CLC

18

Das Carry-Flag wird 0 gesetzt.

SEC

38

Das Carry-Flag wird 1 gesetzt.

CLD

D8

Das Dezimal-Mode-Flag wird 0 gesetzt. Es wird in Hex gerechnet.

SED

F8

Das Dezimal-Mode-Flag wird 1 gesetzt. Es wird in Dezimal gerechnet.

CLI

58

Das IRQ-Disable-Flag wird 0 gesetzt.

SEI

78

Das IRQ-Disable-Flag wird 1 gesetzt.

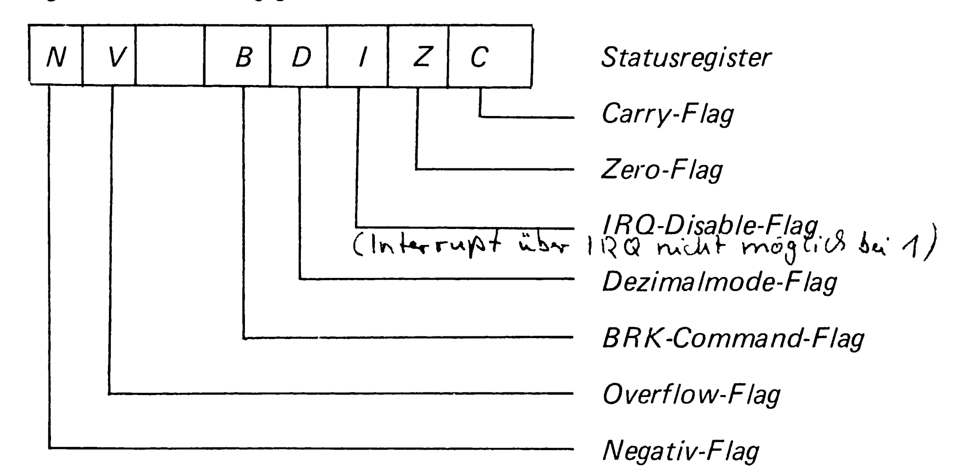
Kein Interrupt über IRQ möglich!

CLV

B8

Das Overflow-Flag wird 0 gesetzt.

Folgende Zuordnung gilt:



PHA

48

Der Inhalt des Akkumulator's wird in die von Stackpointer angegebene Adresse gespeichert. Der Stackpointer wird um 1 heruntergezählt. Der Stackpointer hat folgendes Format

Stack

7 0

1 XXXXXXXX

Die 1 ist vorgegeben.

z.B. Akkuinhalt sei z. B. BB
Stackpointerinhalt sei z. B. FC
Akkuinhalt BB wird also in Adresse 01 FC gespeichert. Anschließend ist Stackpointerinhalt FB.

PHP

08

Der Inhalt des Prozessorstatusregister's wird in die vom Stackpointer angegebene Adresse gespeichert. Der Stackpointer wird um 1 heruntergezählt. Der Stackpointer hat folgendes Format

```
   7              0
 1|X X X X X X X X|
```

Die 1 ist vorgegeben.

z.B. Nach einer Poeration sei der Statusregisterinhalt B0; Stackpointerinhalt sei z. B. AA
d. h. Statusregisterinhalt B0 wird in Adresse 01AA gespeichert.
Nach der Befehlsausführung wird im Stackpointer A9 stehen.

PLA

68

Diese Instruktion benutzt nicht das C- oder 0-Flag. Das N-Flag wird gesetzt wenn der Akkuinhalt in Bit 7 gesetzt ist. Ist Bit 7 0, so wird es zurückgesetzt auf 0. Ist der Akkuinhalt aus der PLA-Instruktion 0 so wird das Z-Flag gesetzt. Jeder andere Inhalt setzt das Z-Flag zurück.

z.B. Stackpointerinhalt sei z. B. 03
Es wird 1 dazuaddiert, ergibt 04. Der Inhalt der Adresse 0104 sei z. B. FF, somit wird FF im Akku geladen, das N-Flag ist auf 1 gesetzt.

Erklärung zu Adresse 0104:

Die Adresse 0104 setzt sich aus dem Stackpointer zusammen. Da der Stackpointer ein Doppelwort ist wobei 1. Wort mit 00000001 festgelegt ist, kommt diese Adresse zustande.

Der Inhalt der Zelle, auf den der Stackpointer zeigt, wird in den Akkumulator gebracht. Danach wird der Stackpointer um Eins erhöht.



28

Diese Instruktion beeinflußt also alle Flags im Statusregister.

z.B. Stackpointerinhalt sei z. B. 04
Es wird 1 dazuaddiert; ergibt 05
Der Inhalt der Adresse 0105 sei z. B. FE. Somit wird FE im Statusregister geladen. Alle Flags außer dem C-Flag sind gesetzt.

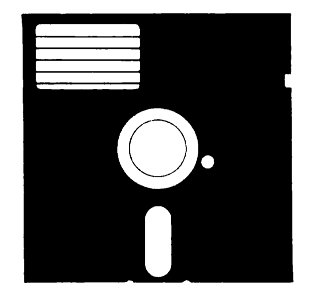
Randbemerkung:
Wenn der Inhalt der Adresse 105 z. B. 00 wäre, so wäre der Inhalt des Statusregister's nur während dem Programmablauf 0. Sie können die 0 aber nicht einsetzen, da Sie dazu einen Break-Befehl eingeben müßten. Ein Break-Befehl einerseits aber setzt das BRK-Flag auf 1, womit der Wert der Anzeige verändert wäre.

Der Inhalt der Zelle, auf den der Stackpointer zeigt, wird in den Akkumulator gebracht. Danach wird der Stackpointer um Eins erhöht.

Quellennachweis und Literaturverzeichnis

Commodore 64 Programmers Reference Guide, Commodore USA
How To Program Your ATARI in 6502 Machine Language, Hofacker Verlag
Programmieren in 6502 Maschinensprache, E.Floegel, Hofacker Verlag
Besonderer Dank gilt Ekkehard Floegel und Franz Ende fuer die Hilfe, sowie Christoph Goethe fuer die Uebersetzung einiger Teile aus dem englischen.

Alle Programme aus diesem Buche auf Diskette



Notizen

Notizen

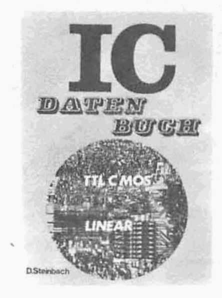
Hofacker-Bücher Deutsch



TBB-Transistor-, Berechnungs- und Bauleitungs-HB, Band 1, Hofacker
Völlig neu überarbeitete Auflage. Das Buch soll bei der täglichen Arbeit im Labor, in der Werkstatt oder am Elektronik-Hobbytisch Ihnen ein guter Begleiter sein. Berechnungsgrundlagen, Berechnungsbeispiele, Tabellen, Vergleichslisten, Digitaltechnik, Netzgeräte, BASIC-Programme zur Berechnung spezifischer Schaltungen usw. sind in übersichtlicher Form dargestellt. Ca. 300 Seiten.
Best.-Nr. 1 **29,80 DM**



Electronic im Auto, H. Gebauer
Mit Handbuch für Polizeiradar. Ein Buch für jeden technisch interessierten Autofahrer. Es zeigt Ihnen die vielen Möglichkeiten zur Verbesserung von Sicherheit, Leistung und Fahrkomfort in Ihrem Auto. Thyristorzündung, Beschleunigungsmesser, Drehzahlmesser, Batterieladegerät, Alarmanlagen u. v. a.
Best.-Nr. 3 **9,80 DM**



IC-Datenbuch, D. Steinbach
Daten- und Auswahllisten der gebräuchlichsten integrierten Schaltkreise. Digital und analog. Gerade bei ICs ist es wichtig die Anschlußfolgen genau zu kennen. Die wichtigsten TTL-Schaltkreise, NF-Verstärker, C-Mos Serie, lineare Schaltungen wie Operationsverstärker, Komparatoren, Spannungsregler, Trigger-Schaltungen, u. v. a. Das IC-Datenbuch wird auch Ihnen ein unentbehrlicher Begleiter bei allen Arbeiten mit integrierten Schaltungen sein.
Best.-Nr. 5 **9,80 DM**



TBB-Handbuch, Band 2, Hofacker
Dieses Buch ist die Fortsetzung des erfolgreichen Handbuches, Band 1. Ein Buch, das sich in der Hand des Praktikers bestens bewährt hat. Weitere neueste Schaltbeispiele und Berechnungsgrundl., Experimentier- und Versuchsbeschreibungen. Integrierte Spannungsregler, Wärmeableitung, Operationsverstärker Einführung, RC-Zeitglieder, Transistortester u. v. a.
Best.-Nr. 2 **19,80 DM**



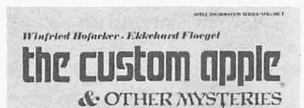
IC-Handbuch, C. Lorenz
Ein Handbuch für digitale und lineare integrierte Schaltungen. Daten- und Auswahl-, Vergleichslisten, Gehäuseformen, Grundlagen, viele Schaltbeispiele, Printvorlagen, u. v. a. Alles über TTL-Technik, C MOS, MOS-Schaltungen, integrierte NF-Verstärker u. v. a.
Best.-Nr. 4 **19,80 DM**

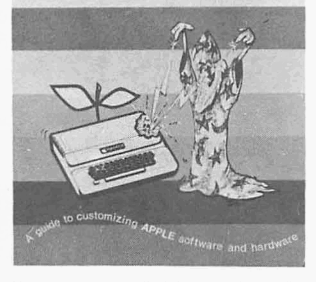


IC-Schaltungen, D. Steinbach
Hier finden Sie eine gelungene Zusammenstellung der wichtigsten Anwendungsbeispiele aus dem Bereich der integrierten Schaltungen. TTL, C MOS, Linear. Alle Schaltungen sind übersichtlich und klar dargestellt und mit einer kurzen, jedoch sehr genauen Beschreibung versehen. Tastenenprellung, Zähler, Impulsgeber, Codierer, Dekodierer, Datenübertragung, Serien-Parallel-Wandler, Digitalvoltmeter u. v. a.
Best.-Nr. 6 **19,80 DM**

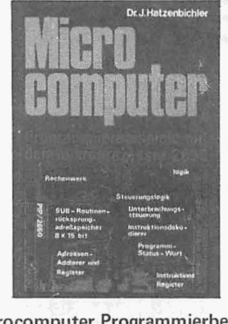


57 Programme in BASIC, Lorenz
Ein Buch mit techn.-wissenschaftlichen Programmen u. einer großen Anzahl von Spielprogrammen in BASIC (Games). Ein Buch für jeden, der sich mit dem faszinierenden Hobby der Mikrocomputertechnik befassen will. Alle Listings sind in BASIC und können auf den meisten Personal Computer Systemen gefahren werden.
Best.-Nr. 31 **39,00 DM**





The Custom Apple & other Mysteries,
von W. Hofacker und E. Flögel
Dieses Buch bringt auf 190 Seiten Großformat eine Vielzahl von Erweiterungsprojekten für Ihren APPLE II. 6522 Ein-/Ausgabekarte, Tonerzeugung mit AY-3-8912, Analog/Digital-Wandler, EPROM-Burner, Anschluß des 8253 von Intel an den APPLE II, Schrittmotorsteuerungen mit APPLE II in BASIC, PASCAL und FORTH. Zu den einzelnen Projekten sind die notwendigen Platinenvorlagen sowie die Software im Buch enthalten. Alle Platinen zum Buch können vom Hofacker Verlag sofort ab Lager bezogen werden.
Best -Nr. 680 (englisch) 79,00 DM



Mikrocomputer Programmierbeispiele für 2650, Dr. J. Hatzenbichler
Eine Einführung in die Programmierung von Mikrocomputern anhand des Prozessors 2650 von Signetics. Viele Programmierbeispiele in Maschinensprache, die Sie auf einem preiswerten Mikroprozessorsystem MIKIT 2650-P2 ausführen können. Zeitschleifenprogr., Blinkschaltung, Lauflicht, Stufenzähler, Stopuhr, Reaktionszeittester, u. a. Zu diesem Buch ist auch ein komplett aufgebautes und getestetes Mikrocomputersystem erhältlich, auf dem Sie alle beschriebenen Programme selbst ausführen können. Über 120 Seiten. (Nur solange Vorrat reicht, wird nicht mehr nachgedruckt.)
Best.-Nr. 33 **19,80 DM**



TINY BASIC Handbuch, Hermann
Das erste deutschsprachige Handbuch über Tom Pittman's TINY BASIC. Eine Einführung in die TINY BASIC-Programmiersprache. Wie kann ich meinen Computer (KIM-1) erweitern und BASIC programmieren. Systemvorschläge. Viele Programmierbeispiele, Tricks und Kniffe.
Best.-Nr. 34 19,80 DM



Der freundliche Computer – Was können Sie mit einem Personal Computer anfangen? T. Munnecke
Das Buch soll Ihnen auf die im Titel gestellte Frage eine ausführliche Antwort geben. Es eignet sich für alle, die bisher viel über Mikros gehört haben und gerne ausführlicher Bescheid wissen möchten. Viele interessante Fakten – Welche Computersprache ? Welche Anwendungen ? Welches Gerät soll ich mir kaufen ? 153 Seiten.
Best.-Nr. 35 **29,80 DM**

ohne Abbildung

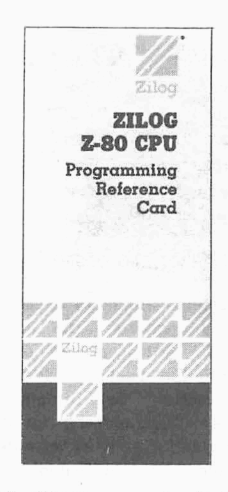
Microcomputer und Roboter
Ein Buch für denjenigen, der sich externe Schaltungen für Microcomputer bauen möchte, die roboterartige Funktionen ausführen können. Spracherkennung, Analog-/Digital-Wandler, Digital-/Analog-Wandler, Ultraschallsensoren, Lichtschranken, Tonerzeugung u. v. a. Erscheint Anfang 1982.
Best.-Nr. 36 **29,80 DM**

ohne Abbildung

Oszillographen Handbuch
Ein Buch für jeden, der seinen Oszillographen optimal nutzen will. Der Anfänger, der noch nie einen Oszillographen benutzt hat, wird auf einfache Weise mit der Technik und Handhabung vertraut gemacht. Auch die Anwendung in Zusammenhang mit den modernen Mikrocomputersystemen wird beschrieben. Oszillograph als alphanumeres Darstellgerät u. v. mehr. Erscheint ca. Anfang 1982.
Best.-Nr. 103 19,80 DM



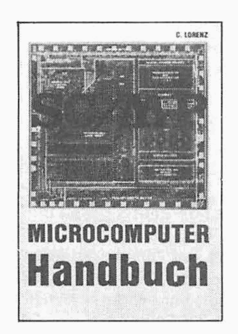
Z80 Assembler Handbuch
Dieses Handbuch braucht jeder, der ein Z80 Computersystem mit Z80 CPU hat (TRS-80, NEC, ALTOS, Sinclair, Video Genie usw.). Auf über 420 Seiten wird jeder Maschinenbefehl erklärt.
Best.-Nr. 8029 29,80 DM



Z-80 CPU Programming Reference Card
Eine praktische und kompakte Zusammenfassung der Z-80-Befehle, Z-80 Pinout, Registeraufbau, ASCII Zeichensatz kompletter Befehlssatz.
Best.-Nr. 252 5,00 DM



CMOS-Experimentierbuch
Eine kleine Einführung in die CMOS Schaltungstechnik. CMOS Grundlagen, Behandlungshinweise für CMOS Bausteine, Zusammenst. der wichtigsten CMOS Bausteine und deren Anschlußbilder. Viele praktische CMOS Schaltbeispiele. Ideal für jeden Elektroniker.
Best.-Nr. 106 1 5,00 DM



Handbuch für SC/MP
Ein echtes Handbuch für SC/MP (INS8060) Microcomputerbesitzer und solche die es werden wollen. Komplette Einführung mit vielen Schaltbeispielen, Schaltbildern und Programmlistings in Maschinensprache und TINY BASIC. Komplettes ELBUG-Listing, CPU-Karte, RAM-Karte, ROM-Karte, ROM-Programmierer, Cassetten-Interface, Fernsehinterface, Netzteil, Hex Ein-/Ausgabe, SC/MP Einkartenmicrocomputer u. v. a. 330 Seiten.
Best.-Nr. 108 29,80 DM



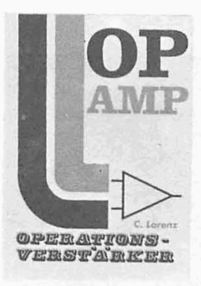
6502 Microcomputer Programmierung, P. Heuer
Eine deutschsprachige Einführung in die Maschinensprachenprogrammierung anhand des 6502 Microcomputers. Ein echtes Anleitungsbuch zum Einstieg in die Microcomputertechnik mit Hilfe des KIM-1. Viele Programmierbeispiele, die von einem Pädagogen speziell für Anfänger entwickelt wurden. Auch PET, AIM, SYM und ATARI-Besitzer brauchen dieses Buch.
Best.-Nr. 109 29,80 DM



Programmierhandbuch für PET
C. Lorenz
Ein für den PET produziertes Buch. Es beginnt da, wo Ihr mitgeliefertes Handbuch aufhört. Einführung Maschinenprogrammierung, Assembler, Ein-/Ausgabeprogrammierung, Programmiertricks, Analog / Digital-Wandler, Graphik, Spracherkennung u. v. a. mehr. Viele Listings, die Sie selbst eintippen können. Viele Informationen eignen sich auch für VC–20-Besitzer. 324 Seiten.
Best.-Nr. 110 29,80 DM



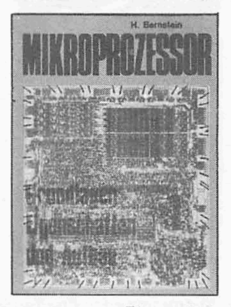
IC Experimentier Handbuch -IC, -EX, C. Lorenz
Eine sehr umfangreiche Schaltungs- und Bauanleitungssammlung mit neuen, jedoch meist beim Fachhandel erhältliche Standard ICs. Rechnerschaltungen, Mikroprozessoren, I/O-Schaltungen, Stoppuhren, druckende und anzeigende Rechner, Digitalvoltmeter, Hilfschaltungen für den Elektronik Experimentierer, A/D-Wandler, Frequenzzähler u. v. a. Viele Schaltungen können auf der IC KIT Experimentierplatine WH-1g aufgebaut werden.
Best.-Nr. 19 19,80 DM



Operationsverstärker, C. Lorenz
Dieses Buch umfaßt das gesamte Gebiet der linearen Schaltungstechnik und stellt ein in dieser Preislage bisher noch nie dagewesenes Nachschlagwerk und Einführungshandbuch dar. Bestens geeignet für das Selbststudium. Nach einer pädagogisch geschickt gemachten Einführung folgen theoretische Arbeitsunterlagen und die zugehörigen Schaltbeispiele mit Daten und Gehäuseanschlüssen. Dieses wertvolle Buch dürfte seinen Platz auch bei Ihren Arbeitsunterlagen finden, und wird dann immer von Nutzen sein, wenn es um die Lösung von nicht routinemäßigen Aufgaben geht. Über 150 Seiten.
Best.-Nr. 20 19,80 DM



Digitaltechnik Grundkurs, C. Lorenz
Ein Einführungskurs in die Digitaltechnik für Anfänger und Fortgeschrittene. Ein Fachbuch für den programmierten Selbstunterricht. Der ideale Kurzlehrgang für das Selbststudium. Der Kurs vermittelt Ihnen alle wichtigen Grundkenntnisse vom TTL-Gatter bis zum Mikroprozessor und Lösung von Schaltungsaufgaben durch Software. Viele Versuchsaufbauten u. Experimente aus diesem Kurs können auf der IC-KIT Platine WH–1g durchgeführt werden. Grundlagen, Gatter, Zähler, programmierbare Zähler, IC-Tester, Schieberegister, Speicher, Mikroprozessoren u. v. a.
Best.-Nr. 21 19,80 DM



Mikroprozessoren, Eigenschaften u. Aufbau, Teil 1, H. Bernstein
Grundlagen, Eigenschaften u. Aufbau von Mikroprozessoren. Organisation von Recheneinheiten und Mikroprogr. Programmierung und Klassifizierung v. Mikroprozessoren. Ablaufdiagramm, Flußdiagramm. Ein Cip-Technik und Multi Chip-Technik, Transfer- und Sprungfunktionen. Speichertechnik: RAMs ROMs, FIFO, FILO. Programmierbare logische Arrays (PLA). Anwendungsbeispiele u. Anwendungsbereiche. Über 120 Seiten.
Best.-Nr. 22 19,80 DM



Elektronik Grundkurs, C. Lorenz
Eine leichtverständliche und pädagogisch geschickt gemachte Einführung in die Technik der elektronischen Schaltungen. Ein Kurzlehrgang und Schnellkurs zugleich. Aber auch ein recht brauchbares Nachschlagwerk für den fortgeschrittenen Elektroniker. Mit wenig Mühe können Sie sich hier die Grundkenntnisse der elektronischen Schaltungspraxis aneignen. Das Buch schafft die Voraussetzungen für ein erfolgreiches und sicheres Arbeiten mit interessanten Schaltkreisen modernster Technologien. Unentbehrlich f. das Experimentieren mit den heutigen modernen hochintegrierten Schaltkreisen.
Best.-Nr. 23 9,80 DM

ohne Abbildung

Programmieren in Maschinensprache mit Z-80 – Band II, Dr. Schmitter
Dieses sehr interessante Werk geht insbesonders auf TRS-80 und Video Genie ein. Es kann jedoch grundsätzlich auf für alle anderen Personalcomputer, die auf der Z80 CPU basieren, verwendet werden (Color Genie, ZX-81, Spectrum, Osborne, Sharp, usw.). Aus dem Inhalt: μC, μP, RAM, ROM - eine Einführung, Unsere ersten Programmierschritte, Schleifen, Flaps und Sprünge, PUSH, POP, [illegible] und Unterprogramme, [illegible]-/Output-Programmierung [illegible] Cassettenport. Arithmetik, relative Sprünge und logische Verknüpfungen. BASIC und Maschinencode, wichtige Hilfsmittel wie Disassembler und Assembler.
Best.-Nr. 24 29,80 DM

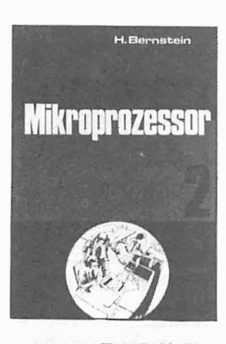
Achtung:
Unter Best.-Nr. 24 erschien bis vor kurzem das Buch Microcomputer-Technik von Blomeyer. Dieses ist vergriffen und wird nicht mehr neu aufgelegt.

ohne Abbildung

68000 Microcomputer Einführung
Eine leicht verständliche Anleitung zur Programmierung des leistungsfähigen 16 Bit Microcomputers von Motorola.
Erscheint ca. Mitte 1983.
Best.-Nr. 25 39,00 DM

Achtung:

Lieber Leser,
die Bestell-Nr. 25 und 27 waren früher Hobby Computer Handbuch (25) und Software Handbuch (27). Diese Titel sind vergriffen und werden vorerst nicht mehr neu aufgelegt.



Mikroprozessor, Teil 2, H. Bernstein
Die Fortsetzung von Best.-Nr. 22. Technologie von Mikroprozessor- und Speicherbausteinen. Festwertspeicher, PROM, EPROM, FIFO, Schieberegister, MPR-, ARL- und SAR-Register. Aufbau eines Mikroprozessorsystems mit 8080, RAM- und ROM-Schnittstellen. Befehlssatz 8080. Über 120 Seiten.
Best.-Nr. 26 19,80 DM



BASIC-M für Motorola EXORset®
Anwender Handbuch
Eine ausgezeichnete und professionelle Einführung in die BASIC-Programmiersprache. Ideal für die Industrie, aber auch ein ausgezeichnetes Werk für den Anfänger. Diese BASIC-Einführung mit vielen Beispielen gehört zu den besten deutschsprachigen Werken auf diesem Gebiet. Der Stoff geht auf den BASIC-M - Dialekt ein, ist aber auch auf jeder andere BASIC-Version anwendbar.
Best.-Nr. 27 29,80 DM



Microcomputer Lexikon u. Wörterbuch von A – Z, C. Lorenz
Einglisch/Deutsch – Der Fachausdruck wird übersetzt, ausführlich erklärt und erläutert. Deutsch/Englisch – Übersetzung des Fachausdrucks. Ein Hilfs- und Arbeitsbuch für jeden, der sich heute mit der modernsten Elektronik beschäftigt. Viele engl. Ausdrücke werden heute in der Elektronik, Computer- und Mikroprozessortechnik verwendet und oft fehlt uns eine genaueste und präzise Erläuterung. Ein Lexikon und Wörterbuch in einem einzigen Buch vereint.
Best.-Nr. 28 29,80 DM



Microcomputer Datenbuch - (englisch) – ELCOMP
Endlich es es da ! Viele unserer Leser haben seit langem auf dieses Buch gewartet. – Und es hat sich gelohnt. Auf über 800 Seiten haben wir hier für Sie die wichtigsten (fast alle) Bauelemente zusammengestellt (Daten, Anschlußbilder etc.), die sich in den heutigen Personalcomputern befinden. Wir haben alle wichtigen Systeme durchforstet und die Bauelemente herausgesucht. TTL, CMOS, Linear, Spannungsregler, CPU-Schaltkreise, 6502, Z80, 8080, 8085, 8086, 1802, 2650, Z8 u. v. a.
Best.-Nr. 29 49,– DM

ohne Abbildung

Floppy Disk Selbstbau-Handbuch
von E. Flögel
Dieses Buch soll Ihnen auf einfachste Weise erklären, wie Sie Ihren Personalcomputer mit möglichst geringen Kosten mit einer Floppy Disk Station ersetzten können. Genaue Bauanleitung mit Software, Hinweise und Tips. Multiuser und Mutlitasking macht die Sache ganz besonders interessant.
Erscheint ca. Ende 1983
Best.-Nr. 30 49,00 DM

Achtung:
Unter Best.-Nr. 30 wurde früher ein Buch "Aktivtraining" angeboten. Dieses ist vergriffen und wird nicht mehr neu aufgelegt.



Elektronik Schaltungen, Hofacker
Die ideale Schaltungssammlung zum Basteln u. Experimentieren, Schaltungen mit Operationsverstärkern, Spannungsreglern, TTL, C-MOS-Schaltkreisen. MOS Uhr mit Wecker-elektronischer Würfel u. v. a.
Best.-Nr. 7 9,80 DM



IC Bauanleitungs Handbuch -IC -KIT, C. Lorenz
Ein Bauanleitungsbuch mit vielen hochinteressanten Bauanleitungen aus dem Bereich der LSI Schaltungstechnik. Schaltbeispiele mit Printvorlagen zum Selbstherstellen der Leiterplatten mit genauesten Beschreibungen. Hochaktuell und brandneu: Funktionsgenerator XR 2206, MOS-Uhr mit Wecker, programmierbarem Wecktongenerator, Schlummerautomatik, IC-Netzteil, Experimentieranleitung und Grundkurs über Flip Flops, u. v. a. Zu allen Schaltungen finden Sie Platinenvorlagen oder Sie können die Experimentierschaltungen auf der Experimentierplatine WH-1 g durchführen. Über 100 Seiten.
Best.-Nr. 8 19,80 DM



Feldeffekttransistoren, C. Lorenz
Der Feldeffekttransistor (FET) gehört heute zu den interessantesten Bauteilen überhaupt. Wie man damit experimentiert, wie man seine Funktion versteht und wie man damit brauchbare u. hochinteressante Schaltungen aufbauen kann, zeigt Ihnen dieses Buch. Grundlagen, Kennlinienfelder, Tabellen, Rechenbeispiele, Anschlußbilder und eine Vergleichsliste für Feldeffektransistoren bilden den Kern dieser umfangreichen Darstellung. Alles in allem finden Sie hier eine praxisnahe und komplette Arbietsunterlage, mit der Sie im Beruf und auch im Hobby erfolgreich arbeiten können.
Best.-Nr. 9 9,80 DM



Elektronik und Radio, C. Lorenz
4. Auflage. Völlig neu bearbeitet und stark erweitert. Eine Schritt für Schritt Einführung in die Radiotechnik mit vielen Bildern. Vom einfachen Diodenempfänger (Detektor) bis zu interessanten Sender- und Empfängerschaltungen (Minispione). IC-Radio, IC-Sender, Antennen, Berechnungsgrundlagen, Tabellen u. v. a. Über 150 Seiten.
Best.-Nr. 10 19,80 DM



NF-Verstärker, C. Lorenz
Grundlagen der integrierten NF-Verstärker, Berechnung von kompletten IC-NF-Verstärkerstufen. Anwendungsbeispiele mit den interessantesten und gebräuchlichsten Standard IC-NF-Verstärkern wie TBA 800, TBA 830, usw. Printvorlagen, Auswahltabellen, Experimentieranleitungen und Anschlußbilder machen dieses Buch zu einem unentbehrlichen Begleiter für alle, die sich m. NF-Verstärkern beschäftigen wollen.
Best.-Nr. 11 9,80 DM



BIS, Beispiele integrierter Schaltungen, H. Bernstein
Auf über 130 Seiten Anwendungsbeispiele mit integrierten Schaltkreisen, Zeitgeber 555, Funktionsgenerator ICL 8038, Opto Elektronik, Operationsverstärker, Festwertspeicher (ROM), u. v. a.
Best.-Nr. 12 19,80 DM



HEH, Hobby Elektronik Handbuch C. Lorenz

Das Schaltungsbuch f. jeden Hobbyelektroniker, Schaltbeispiele und Bauanleitungen aus dem gesamten Hobbybereich. Lichtorgeln, Alarmanlagen, Eiswarngerät fürs Auto, PLL-Schaltungen u. v. a.

Best.-Nr. 13 9,80 DM



Viel mehr als 33 Programme für den Sinclair SPECTRUM, R. G. Hülsmann

Ein echt aufregendes Buch für jeden SPECTRUM-Besitzer. Viele Tricks und Tips. 33 Superprogramme wie 3D-Graphik, Crazy Kong, Musik-Computer. Unterprogramme in Maschinensprache, zehn kurze Progr. für den Spectrum mit 16K RAM wie Mondlandung, Spielautomat, Irrgarten, Todeshöhle usw. Sie werden begeistert sein!

Best.-Nr. 144 29,80 DM



Opto-Handbuch, C. Lorenz

Das Handbuch für die gesamte Optoelektronik. Eine Einführung und ein ideales Nachschlagwerk. Grundlagen, Definitionen aller Kenngrößen, Opto-Lexikon, Berechnungsgrundlagen, Lichtsender, Lichtempfänger, Anzeigen, Infrarot Detektoren, Optokoppler, Opto-Vergleichsliste, u. v. a.106 Seiten.

Best.-Nr. 15 19,80 DM



C MOS Einführung, Entwurf, Schaltbeispiele, Teil 1 H. Bernstein

Vom C MOS Gatterbaustein über Schieberegister und Zähler bis hin zum C MOS Schreib- Lesespeicher. Insgesamt werden neunzehn interessante und bekannte C MOS Schaltkreise beschrieben. Zu jedem Bauelement sind genaue Daten, Schaltbild und Anwendungsbeispiele angegeben. Im großen Applikationsteil finden Sie: C MOS-Kippstufen, Addierwerke u. Rechenschaltungen, Digital Analog Wandler, Schieberegister für analoge Spannungen, Multiplexsysteme f. analoge Signale u. v. a. Eine komplette Einführung u. gut geeignet für das Selbststudium der C MOS Technik. 140 Seiten.

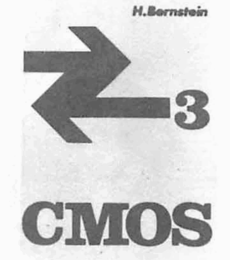
Best.-Nr. 16 19,80 DM



C MOS Entwurf u. Schaltbeispiele, Teil 2, H. Bernstein

Fortsetzung von Best-Nr. 16. Anwendungsbeispiele mit genauen Schaltungsbeschreibungen und Bauelementunterlagen. Daten, Anschlußbelegungen weiterer wichtiger hochintegrierter C MOS Elemente. Ein komplettes Arbeits- u. Experimentierbuch. C-MOS Uhrenschaltungen, Schieberegisterschaltungen, Parallel-Serien Umsetzung, statische u. dynamische Speicherschaltungen, Zählschaltungen, Digital Analog-Wandler, Analog Digital Wandler. Digital Voltmeter, I/O Registerschaltungen. RAM und ROM Anwendungen. Über 140 Seiten.

Best.-Nr. 17 19,80 DM



C MOS Entwurf u. Schaltbeispiele, Teil 3, H. Bernstein

Fortstetzung von Best.-Nr. 17. Eine sehr umfangreiche Applikationssammlung mit hochintegrierten C MOS Elementen. Speicher- und Steuerschaltungen, Multiplex- und Datenbussysteme, Liquid Cristal Anzeigen, Uhrenschaltungen, PLL-Schaltungen, Optoelektronik in Verbindung mit C MOS. Aufbau und Wirkungsweise der Prozeßrechentechnik, Arithmetische Logische Einheiten (ALU) u. andere wichtige Funktionen aus der Prozeßrechentechnik. RAMs, ROMs, und FIFO-Speicherschaltungen.

Best.-Nr. 18 19,80 DM



Programmieren mit TRS-80, Stübs
Das erste in einem deutschen Verlag produzierte Buch über den erfolgreichen Personal Computer von TANDY. Ein Buch für jeden, der einen TRS-80 bereits besitzt oder vor der Entscheidung steht, welchen Computer er sich anschaffen soll. Einführung, Programmiertricks, Erweiterungen, Maschinenprogrammierung und viele Programme (Listing mit Beschreibung). 202 Seiten.
Best.-Nr. 111 29,80 DM



BASIC Programmierhandbuch
Einführung und Nachschlagewerk. Speziell für die BASIC-Versionen der modernen Microcomputersysteme. Jeder Befehl wird ausführlich beschrieben und ein Beispielprogramm gezeigt. Sehr übersichtlich und praktisch. Am Schluß finden Sie ein komplettes BASIC-Programm, das Ihnen über einen Computer BASIC lehrt.
Best.-Nr. 113 19,80 DM



Der Microcomputer im Kleinbetrieb
Das Buch für jeden Geschäftsmann. Auf über 170 Seiten erfahren Sie, was Sie als Gewerbetreibender oder freiberuflich Tätige über Microcomputer und die Anwendung wissen sollten. Geschichtlicher Hintergrund Geräteauswahl, Beispiele aus der Praxis, Programmbeispiele wie z. B. Textverarbeitung, Reisebüro, Ladenkasse, Adressverwaltung, u. v. a. Betriebswirtschaftliche Auswertung, Finanzbuchhaltung, Erfolgsanalyse mit dem Microcomputer, Liquiditätsrechnung, kurzfristige Erfolgsrechnung, Microcomputer für Freiberufler, Grundlagen der Finanzbuchhaltung für Microcomputeranwender. Dieses Buch kann Ihnen als Geschäftsmann für die Zukunft tausende einsparen.
Best.-Nr. 114 39,80 DM



PASCAL Handbuch, E. Flögel
Von BASIC zu PASCAL. Ein Einführungs- Lehr und Arbeitsbuch für jeden der sich mit PASCAL beschäftigen will oder muß. Viele Programmbeispiele, viele Tricks wie PEEK und POKE, Einbinden von Maschinenprogrammen u. v. a.
Best.-Nr. 112 29,80 DM



16 Bit Microcomputer, J. Koller
Einführung, Daten, Eigenschaften, Anwendungen. Dieses Werk ist eine echte Sensation !
Alle 16 Bit Prozessoren werden beschrieben und erläutert. Applikationsbeispiele, Programmierhinweise. TMS 9900, 8086, Z8000, MC 68000, NS 16000, IAPX 486, IAPX 432. Über 370 Seiten.
Best.-Nr. 116 29,80 DM

ohne Abbildung

FORTRAN für Heimcomputer
Einführung in die FORTRAN-Programmiersprache mit vielen Beispielen. Grundsätzliches über die verschiedenen Microcomputersysteme, die bereits mit FORTRAN-Compiler lieferbar sind. Allgemeine Übersicht, Tips und Hinweise.
Erscheint ca. Ende 1982 .
Best.-Nr. 117 19,80 DM







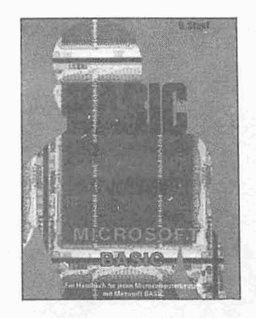
Programmieren in Maschinensprache mit 6502, E. Flögel, W. Hofacker
Das deutschsprachige Werk über 6502 Maschinenprogrammierung. Einführung, Grundlagen, Eigenschaften, Adressierungsarten, Befehlsarten. Wie entwickelt man ein 6502 Maschinenprogramm? Handassemblierung, viele Programmbeispiele mit genauen Angaben direkt zum Eingeben in den Apple II mit Adressangabe (keine blutleeren Beispiele ohne Adresse), Verwendung von Assemblern. AIM-Assembler, Disassembler, Relocator, 6522 VIA, 6520, Interrupt, Fehlersuche in Maschinenprogrammen, Maschinensprache, Programmiertricks. Spezielle Abschnitte für Maschinensprachenprogrammierung über PET, CBM 3000, CBM 4000 u. VC-20, ATARI 400/800, Apple II, AIM sowie Ohio Scientific Challenger. Dieses Buch sollte jeder 6502 Systemanwender besitzen. Ca. 240 Seiten.
Best.-Nr. 118 49,00 DM

BASIC für Fortgeschrittene
Endlich ein BASIC-Buch für den fortgeschrittenen Programmierer. Alle wichtigen Befehle aus der Stringmanipulation, Disk-Befehle, WAIT, INSTR, WHILE WHEND usw. werden an Beispielen besprochen. Alle Befehle sind übersichtlich wie in einem Nachschlagwerk mit großen Überschriften angeordnet. Dann folgt ein umfangreicher BASIC Kurs für Fortgeschrittene mit vielen Beispielen (Inventur, Rechnungen schreiben, Adressenverwaltung usw.). Am Schluß finden Sie dann noch einen Vergleich der wichtigsten Sortiermethoden sowie ein Programm zur Vorhersage von Ereignissen.
Best.-Nr. 122 39,00 DM

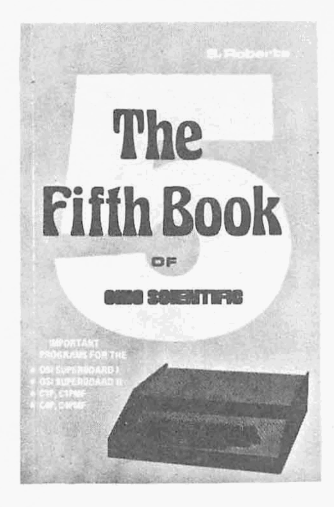
Programmieren in Maschinensprache (Z80), C. Lorenz
Eine sehr ausführliche Einführung in die Z80 Maschinensprache mit vielen Beispielen. Die Beispiele können mit Hilfe des TRS-80 Level II sowie dem T-BUG von TANDY und den T-BUG-Erweiterungen (IN LOCO, T-STEP, T-LEGS) ausgeführt werden. Ein unentbehrliches Buch für jeden, dem die BASIC-Programmiersprache von der Geschwindigkeit her zur Lösung seiner Aufgaben nicht mehr ausreicht.
Best.-Nr. 119 39,00 DM

Anwenderprogramme für TRS-80 von Martin Stübs
Ein Buch, voll mit interessanten Anwenderprogrammen für TRS-80 Level II 16K und Video Genie (teilweise Diskette u./od. Cassette). Hauptsächlich Programme für den Manager, Geschäftsmann, Klein- und Mittelbetrieb. Auch einige interessante Spiele sind enthalten. Terminkalender, Reservierungsprogramm für Omnibusunternehmen und Hotels, Textverarbeitung, usw.
Best.-Nr. 120 29,80 DM





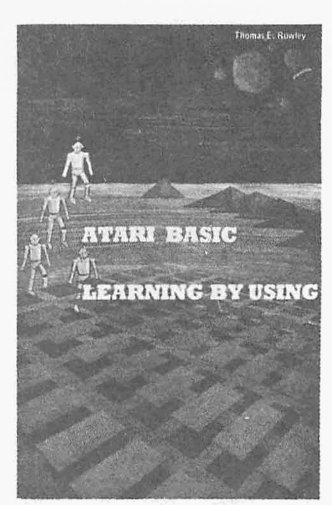
Microsoft BASIC-Handbuch
Die deutsche Übersetzung des erfolgreichen Microsoft BASIC-Handbooks. Leicht verständliche Einführung mit vielen interessanten Programmbeispielen. Das kompetente Werk von Microsoft selbst. Ideal als Zusatzliteratur zu jedem BASIC-Buch.
Best.-Nr. 121 29,80 DM



The Fifth Book of Ohio Scientific
Kein anderer Verlag der Welt hat fünf Bände für den so erfolgreichen und leistungsfähigen Personalcomputer bisher produziert. Der 5 Band bringt wieder eine Vielzahl von sehr interessanten BASIC- und 6502 Maschinenprogrammen. Neben einer Adressenverwaltung, einem Textprogramm, Fakturierprogrammen und Utilities finden Sie auch wieder viele interessante Spiele sowie eine komplette Abhandlung über Start- und Landesimulationen, die sich besonders als Hilfsmittel zur Entwicklung eigener Mondlandespiele eignen. Dem Buch kann vom erfahrenen BASIC-Programmierer auch viel Stoff zur Implementation f. andere Rechner entnommen werden (englisch).
Best.-Nr. 161 19,80 DM



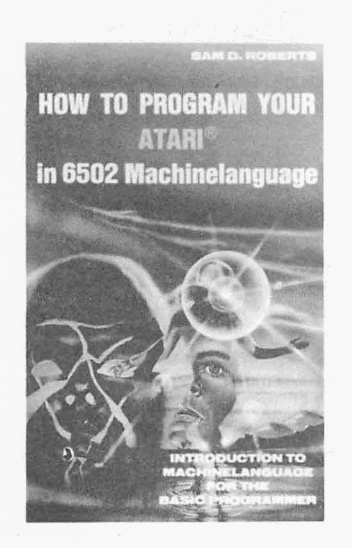
ZX-81 / TIMEX
Programming in BASIC and Machine Language by E. Flögel
Ein Buch mit vielen Tips und Programmen für den erfolgreichen Sinclair ZX-81 Personalcomputer. Dieses Buch ist eine Übersetzung unseres Titels Nr. 140.
Sehr interessant ist die Anleitung zum Aufbau einer Z-80 PIO an den ZX-81.
Best.-Nr. 174 29,80 DM



ATARI BASIC-Learning by USING v. Thomas E. Rowley (engl.)
Ein echtes Action-Buch für Ihren ATARI 400/800. Hier findet mehr statt als nur lesen. Sie verwenden es und machen neue Entdeckungen. Viele nützliche Routinen und Hilfsprogramme. Grafik, Tonerzeugung, Joystickprogrammierung, PEEK und POKE und Special-"Struff".
Best.-Nr. 164 19,80 DM



Programming in 6502 Machinelanguage for PET + CBM
Dieses Buch enthält eine große Auswahl sehr wertvoller Informationen für den PET und CBM Besitzer der 3000er und 2000er Serie. Neben dem Listing und Beschreibung für einen sehr leistungsfähigen EDITOR/Assembler in Maschinensprache mit Beispiel für den Sie einen Disassembler, Linker, Editor, Assembler in BASIC und Maschinencode sowie einen kompletten Maschinensprachenmonitor für PET 2001. (engl.)
Best.-Nr. 166 49,00 DM



How to program your ATARI in 6502 Machinelanguage, Roberts
Eine sehr gute Einführung in die 6502 Maschinensprache mit dem ATARI 400/800.
Im einzelnen leicht verständlich und überschaubaren Lektionen werden Sie an Hand von praktischen Beispielen in die wunderbare Welt der Maschinensprache geführt. Sehr viele interessante Programmbeispiele für den ATARI. Tricks, Kniffe und Tips.
Best.-Nr. 169 29,80 DM

Hiermit gebe ich folgende Bestellung unter Anerkennung
Ihrer Lieferungs- und Zahlungsbedingungen auf.



Menge	Artikelbezeichnung	Einzelpreis

Menge	Artikelbezeichnung	Einzelpreis

Unterschrift (für Jugendliche unter 18 J. der Erziehungsberechtigte) Datum

---

Hiermit bestelle ich die ELCOMP-Sonderhefte bis auf Widerruf.

○ Ich bitte um Abbuchung von DM 29,80 (incl. Versand und Verpackung) von meinem Konto (Giro oder Postscheckkonto)

Kto.-Nr. .... BLZ ....

Geldinstitut: .... Ort: ....

○ Ich bitte um Lieferung per Nachnahme. Mit der Lieferung eines Heftes wird der Betrag von DM 29,80 + DM 6,50 NN-Gebühr erhoben.

Name/Vorname

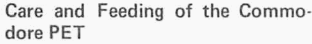
Straße/Haus-Nr.

PLZ Ort

Datum................ Unterschrift ....

ELCOMP-Bücher Englisch





Care and Feeding of the Commodore PET

Das ideale Buch für den Hardware-Bastler. Viele Tricks, Schaltbilder, Hinweise und Erläuterungen für den, der gerne selbst Erweiterungen bauen möchte. Memory Map für 8k PET und CBM, Bauanleitung für eine serielle Schnittstelle u. v. a.

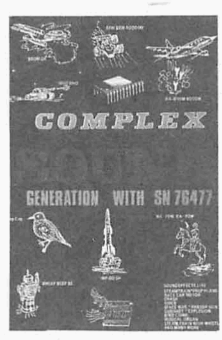
Best.-Nr. 150 19,80 DM



Microsoft 8k BASIC Reference Manual

Eine sehr gute BASIC-Einführung. Auch als Handbuch zum Nachschlagen bestens geeignet. Ideal für jeden PET, CBM, TRS-80, KIM-BASIC, SYM-BASIC, AIM- und APPLE-Besitzer. 73 Seiten DIN A4 mit vielen Beispielen.

Best.-Nr. 151 9,80 DM





Complex Sound Generation with SN 76477

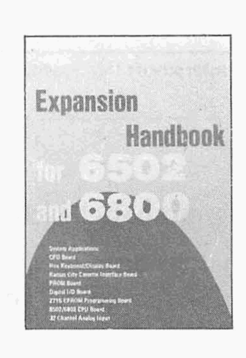
Ein Applikationsheft für einen der interessantesten integrierten Bausteine unserer Zeit. Ein LSI-Baustein zur Tonerzeugung. Je nach äußerer Beschaltung können Sie mit diesem Baustein die verrücktesten Töne erzeugen. Dampfeisenbahngeräusch mit Dampfpfeife, Vogelgezwitscher, Hundegebell, elektronische Orgel, Schuß mit Explosion u. v. a. mehr.

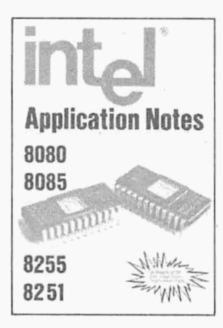
Best.-Nr. 154 9,80 DM

Expansion Handbook for 6502 and 6800

Das ideale Handbuch für alle KIM, SYM, AIM, PET und Challenger Computer-Freunde. Das Buch beschäftigt sich ausschließlich mit dem S-44-Bus. Dies ist exakt der Bus von SYM, AIM und KIM. Sehr viele Schaltbilder: CPU-Platine, Hex-Tastatur Eingabe, Knsas City Interface, RAM u. ROM-Karte, Analog-Eingabe Board u. v. a. Das Buch ist für jeden 6502 Systembesitzer unentbehrlich. Ca. 150 Seiten.

Best.-Nr. 152 **19,80 DM**



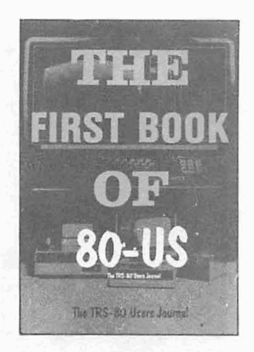


Intel Application Notes (8080, 8085, 8255, 8251)

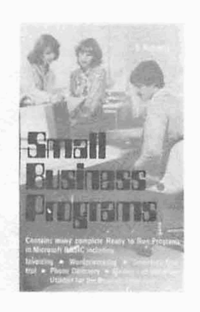
Dieses Buch braucht jeder, der mit 8080, 8085 oder Z-80 Mikroprozessoren arbeitet.

Wir haben die interessantesten Applikationsberichte in diesem Buch zusammgefasst. Aus dem Inhalt: Designing with Intel's Static RAM's 2102, Memory Design with the Intel 2107B. 8255 Programmable Peripheral Interface Applications, Using the 8202 Dynamic RAM Controller u. v. a.

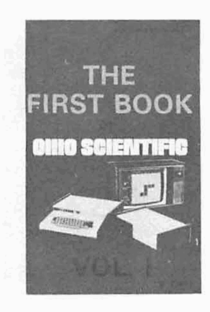
Best.-Nr. 153 **29,80 DM**



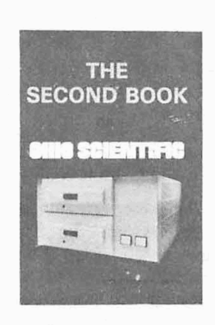
The First Book of 80-US
Für den TRS-80 Freund eine echte Preissensation. Die ersten fünf Hefte aus 80-US Journals in einem Sammelband zusammengefaßt. Voll mit vielen sehr interessanten Hard- und Softwareideen, Tricks. Viele komplette Programmbeispiele (Listings) in BASIC u. Z-80 Maschinensprache. Über 250 Seiten DIN A4. Farbiger Umschlag. Dieses Buch sollte jeder TRS-80 Besitzer oder der es werden will im Schrank haben.
Best.-Nr. 155 — 29,80 DM
(solange Vorrat reicht)



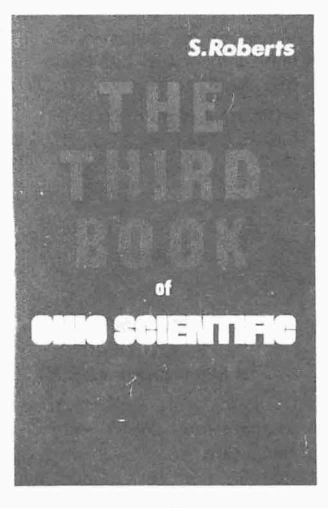
Small Business Programs, S. Roberts
Ein Buch für denjenigen, der die modernen Microcomputer (speziell TRS-80, Apple, PET, North Star, Challenger) zur Rationalisierung in seinem Klein- oder Mittelbetrieb einsetzen möchte. Viele nützliche Tips, Hinweise und Programmierbeispiele. Dieses Buch sollte jeder Geschäftsmann u. Microcomputerfreund besitzen.
Best.-Nr. 156 — 29,80 DM



The first Book of Ohio Scientific
Das erste weltweit produzierte Buch für die erfolgreiche Ohio Scientific Challenger Computerserie. Grundlagen, viele Programmiertricks Hardwaretips, Umbauanleitungen, Programmierbeispiele u. v. a. Glanzumschlag, 186 Seiten.
Best.-Nr. 157 — 19,80 DM
(solange Vorrat reicht)



The second Book of Ohio Scientific
Eingehende Beschreibungen über praktische und geschäftsorientierte Software. Speicher Test Programm, Tricks und Tips für Disketten-Anwender. Mini-Floppy-Expansion u. v. a. 159 Seiten.
Best.-Nr. 158 — 19,80 DM



The third book of Ohio
Wie erweitere ich mein Challenger System ? Universelle I/O-Karte, EPROM-Burner für 2716, EPROM, RAM-Karte, 6522 VIA-Karte. Wo notwendig mit kompletter Software. Dieses Buch braucht jeder Ohio-Benutzer. Komplette Schaltbilder und Aufbauhinweise.
Best.-Nr. 159 — 19,80 DM



The fourth Book of Ohio Scientific
Ein Buch voll mit Programmen für das Superboard, C4P, C4PMF und C28P. Die Softwarequelle für jeden Challenger-Fan. Alle Programme sind getestet und auch auf Cassette verfügbar. 170 Seiten Listings und Beschreibungen.
Best.-Nr. 160 — 29,80 DM
Best.-Nr. 8324 Cassette — 29,80 DM



IEC Bus-Handbuch, M. P. Gottlob
Ein Handbuch und Nachschlagewerk für alle Besitzer von Computern mit IEC (IEEE 488 Bus). Dazu gehört auch der PET sowie alle CBM-Computer. Grundlagen, das BUS-System, Meßdatenübertragung, Adressierung eines Instruments, kl. IEC-BUS-Lexikon u.v.a.
Best.-Nr. 123 19,80 DM



Programmierbeispiele für CBM
Ein Buch mit vielen BASIC-Programmen für CBM und PET. Spiele, Geschäftsbereich, Erziehung und Wissenschaft, Utilities, Hilfen für Maschinensprachenprogrammierung, trickreiche Programme. Viele Programme für wenig Geld.
Best.-Nr. 130 19,80 DM



Programmieren in Maschinensprache mit CBM
An Hand eines praktischen Beispiels (Sortierroutine) wird der Unterschied zwischen BASIC und Maschinenprogrammen gezeigt. Das Maschinenprogramm kann mit dem leistungsfähigen MONJANA/1 Monitor in ROM erstellt werden. Am Schluß finden Sie weitere wichtige Informationen wie Dez/Hex-Umrechnungstabelle, Befehlslisten, ASCII-Tabelle sowie eine ROM-Vergleichsliste zwischen 8k PET und den neuen CBM-Maschinen.
Best.-Nr. 124 19,80 DM

MONJANA Monitor im ROM
Best.-Nr. 1241 79,00 DM

ohne Abbildung

ELCOMP
Fachzeitschrift
für Microcomputertechnik

Die kompetente Fachzeitschrift für das moderne Gebiet der Microcomputertechnik. Erscheint 2 x pro Jahr. Preis pro Heft 29,80 DM incl. MwSt., Porto und Verpackung. Wer die neuesten Informationen aus diesem Gebiet für sich nützen möchte, muß ELCOMP lesen. Software, Technische Tips, Programmiertricks, Bauanleitungen, Systembeschreibungen, u. v. a.
Best.-Nr. 125 29,80 DM



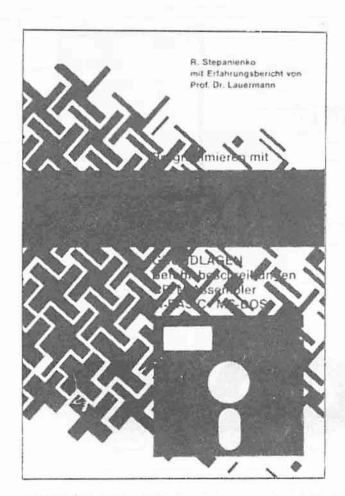
Einführung in die Microcomputer Programmierung mit 6800
Eine sehr gute Einführung in die Microcomputertechnik mit Hilfe des Mikroprozessors 6800. Ausführliche Erklärungen mit vielen Beispielen und Anleitungen. Theoretische Grundlagen. CPU-Architektur, Befehlssatz, Systemaufbau, Hilfsmittel der Programmierung, Trainingsprogramme, Systemkomponenten, FIRMWARE. Ein komplettes Monitorprogramm (Betriebssystem) ist als Listing enthalten. Über 250 Seiten.
Best.-Nr. 127 49,00 DM

ohne Abbildung

Programmieren mit dem CBM
Ein Hand- und Programmierbuch für alle CBM-Besitzer der 3000, 4000 sowie der 8000er Serie. Viele Tricks und Programmierbeispiele, Anleitungen.
Best.-Nr. 128 29,80 DM

ohne Abbildung

ELCOMP Leser Programmierhandb.
Hier fassen wir die besten Programme unserer ELCOMP-Leser zusammen. Programme für PET, CBM, TRS-80, AIM, Superboard, C4P, Exidy, Sharp, MZ80K, Apple II, Nascom I und II und TI 99/4 werden als Listing mit kurzer Beschreibung allen Lesern zugänglich gemacht. Erscheint Anfang 1982.
Best.-Nr. 129 69,00 DM



CP/M Handbuch
Grundlagen, Einführung, Hilfs- und Handbuch für jeden der mit dem "Software-Bus" arbeiten möchte. Ideal auch für Anfänger. Praktisches Handbuch für den Profi. Erscheint ca. Ende 1981.
Best.-Nr. 132 19,80 DM



FORTH Handbuch und Einführung von E. Flögel, W. Hofacker
FORTH ist nicht nur eine sehr leistungsfähige Programmiersprache – es ist schon fast eine "Religion". FORTH eignet sich bestens für industrielle Steuerungen, Grafik etc. Grundlagen und viele Programmbeispiele (Apple II, Ohio, ATARI usw.). Erscheint Mitte 1982.
Best.-Nr. 137 39,00 DM



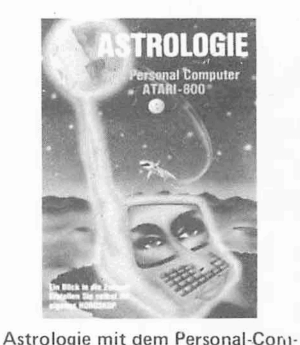
BASIC für blutige Laien, Matthaei
Endlich ein Buch für den Anfänger und Laien. Ziel dies Buches ist es, dem "blutigen Laien" die Grundlage der Programmiersprache BASIC zu vermitteln. Lernen wird nun zum echten Vergnügen und Freizeitspaß. Auch der Preis macht Spaß.
Best.-Nr. 139 nur 19,80 DM



Programmieren in BASIC und Maschinencode mit dem ZX81, E. Flögel
Ein Buch für Sinclair ZX81 Besitzer und solche die es werden wollen. Was heißt Programmieren?, Programmerstellung in BASIC, Spiele, Spielelemente, Programme für die Schule, Datenverwaltungsprogramme, Lagerbestand, Schallplattenverzeichnis, Programmieren in Z80 Maschinensprache, Anschluß einer PIO und externer Schaltungen, Lösen von digitalen Steuerungsaufgaben mit dem ZX81 u. v. a. Dieses Buch gehört auf den Tisch eines jeden ZX81 Besitzers (mehr als 20 komplette Programme, Maschinensprachen-Monitor, etc.).
Best.-Nr. 140 29,80 DM



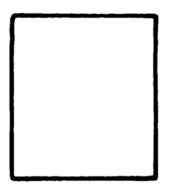
Programme für den VC-20 von W. Hofacker
Ein Buch des Hofacker Verlages speziell für den VC-20 Volkscomputer von Commodore. Viele komplette Programme wie Wortprozessor, Maschinensprachenmonitor, lustige Spiele, Programmieren in Maschinensprache, Ein-/Ausgabeprogrammierung. Wichtige Adressen des Betriebssystems, Tabellen, Speichererweiterungen, Dual-Joystick Bauanleitung u. v. a. Sie werden begeistert sein.
Best.-Nr. 141 29,80 DM



Astrologie mit dem Personal-Computer ATARI 800, W. Hofacker
Ein Blick in die Zukunft. Erstellen Sie selbst Ihr eigenes Horroskop. Dieses Büchlein zeigt dem interessanten Leser wie man sein eigenes Horroskop mit professioneller Genauigkeit berechnen kann. Was braucht man dazu? Wie geht man vor? Was hat es mit der Astrologie auf sich? Was braucht man für die Deutung? Enthält ein komplettes Listing in BASIC und Ma.-Code (ATARI 800, 48K RAM + Disk).
Best.-Nr. 175 49,00 DM

ELCOMP

POSTKARTE



Absender
Bitte deutlich ausfüllen

____
Vorname/Name

____
Beruf

____
Straße/Nr.

____
Plz Ort

ELCOMP

MIKROCOMPUTER BOOK STORE

Tegernseerstr. 18

D-8150 Holzkirchen /Obb.

- -

ABSENDER:

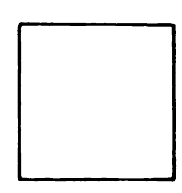
....
Name, Vorname

....
Straße

( ) ....
PLZ Ort

....
Telefon

Die Bestellung der Sonderhefte kann JEDERZEIT widerrufen werden. Die Belieferung wird zum nächst erscheinenden Heft eingestellt.
Eine Kündigung NACH Auslieferung eines Sonderheftes (rückwirkend) ist NICHT möglich.



ELCOMP

Ing. W. Hofacker GmbH
Tegernseer Straße 18
D-8150 Holzkirchen

Notizen

Notizen

Notizen

Notizen

Weitere interessante Bücher von Hofacker:

Best.-Nr.	Titel	Preis/DM
	Bücher in deutscher Sprache aus dem Hofacker-Verlag	
1	Transistor Berechnungs- und Bauanleitungsbuch – 1	29,80
2	Transistor Berechnungs- und Bauanleitungsbuch – 2	19,80
3	Elektronik im Auto	9,80
4	IC-Handbuch, TTL, CMOS, Linear	19,80
5	IC-Datenbuch, TTL, CMOS, Linear	9,80
6	IC-Schaltungen, TTL, CMOS, Linear	19,80
7	Elektronik Schaltungen	19,80
8	IC-Bauanleitungs-Handbuch	19,80
9	Feldeffekttransistoren	9,80
10	Elektronik und Radio	19,80
11	IC-NF Verstärker (i. V.)	9,80
12	Beispiele integrierter Schaltungen (BIS)	19,80
13	HEH, Hobby Elektronik Handbuch	9,80
15	Optoelektronik Handbuch	19,80
16	CMOS Teil 1, Einführung, Entwurf, Schaltbeispiele	19,80
17	CMOS Teil 2, Entwurf und Schaltbeispiele	19,80
18	CMOS Teil 3, Entwurf und Schaltbeispiele	19,80
19	IC-Experimentier Handbuch	19,80
20	Operationsverstärker	19,80
21	Digitaltechnik Grundkurs	19,80
22	Mikroprozessoren, Eigenschaften und Aufbau	19,80
23	Elektronik Grundkurs, Kurzlehrgang Elektronik	9,80
24	Progr. in Maschinensprache mit Z80, Band II	29,80
25	68000 Microcomputer Einführung (i. V.)	39,00
26	Mikroprozessor, Teil 2	19,80
27	BASIC-M Anwender-HB f. 6800/09/68000 (Motorola)	29,80
28	Lexikon + Wörterbuch f. Elektr. u. Mikroprozessor	29,80
29	Mikrocomputer Datenbuch	49,80
30	Floppy Disk Selbstbau-Handbuch (i. V.)	49,00
31	57 Programme in BASIC	39,00
33	Microcomputer Programmierbeispiele	19,80
34	TINY-BASIC Handbuch	19,80
35	Der freundliche Computer	29,80
103	Oszillographen-Handbuch	19,80
108	Rund um den Spectrum (Progr., Tips und Tricks)	29,80
109	6502 Microcomputer Programmierung	29,80
110	Programmierhandbuch für PET	29,80
111	Programmieren mit TRS-80 (Video Genie)	29,80
112	PASCAL-Programmier-Handbuch	29,80
113	BASIC-Programmier-Handbuch	19,80
114	Der Microcomputer im Kleinbetrieb	39,80
115	6809 Programmier Handbuch (i. V.)	49,00
116	Einführung 16-Bit Micorocomputer	29,80
117	FORTRAN für Heimcomputer	19,80
118	Programmieren in Maschinensprache mit dem 6502	49,00
119	Programmieren in Maschinensprache (Z80) Band I	39,00
120	Anwenderprogramme für TRS-80 u. Video Genie	29,80
121	Microsoft BASIC-Handbuch	29,80
122	BASIC für Fortgeschrittene	39,00
123	IEC-Bus Handbuch	19,80
124	Programmieren i. Ma.-Sprache mit Commodore-64	29,80
127	Einführung i. d. Microcmputer-Progr. mit 6800	49,00
128	Programmieren mit dem CBM	29,80
130	Programmierbeispiele für CBM	19,80
132	CP/M-Handbuch	19,80
133	Handbuch für MS/DOS (i. V.)	29,80
137	FORTH Handbuch	49,00
139	BASIC für blutige Laien	19,80
140	Progr. i. BASIC u. Maschinencode mit dem ZX81	29,80
141	Programme f. VC-20 (Spiele, Utilities, Erweiterungen)	29,80
143	35 Programme für den ZX81	29,80
144	33 Programme für den ZX-Spectrum	29,80
145	64 Programme für den Commodore 64	39,00
146	Hardware-Erweiterungen für den C-64 (i. V.)	39,00
147	Beherrschen Sie Ihren Commodore 64	19,80
148	Programmierhandbuch für SHARP	49,00
149	Programme für TI 99/4A	49,00
175	Astrologie auf dem ATARI 800	49,00
8029	Z-80 Assembler-Handbuch	29,80
	Bücher in englischer Sprache	
	1. Von ELCOMP Publishing, Inc., Los Angeles, CA.	
150	Care and Feeding of the Commodore PET	19,80
151	8K Microsoft BASIC Reference Manual	9,80
152	Expansion Handbook for 6502 and 6800	19,80
154	Complex Sound Generation using the SN76477	9,80
156	Small Business Programs	29,80
158	The Second Book of Ohio Scientific	19,80
159	The Third Book of Ohio Scientific	19,80
160	The Fourth Book of Ohio Scientific	29,80
161	The Fifth Book of Ohio Scientific	19,80
162	ATARI Games in BASIC	19,80
163	The Peripheral Handbook (i. V.)	29,80
164	ATARI-BASIC Learning by Using	19,80
166	Programming in 6502 Machinelanguage PET/CBM	49,00
169	How to Progr. your ATARI in 6502 Machinelanguage	29,80
170	FORTH on the ATARI – Learning by Using	29,80
171	See the Future with your ATARI (Astrology)	49,00
172	Hackerbook I (Tricks + Tips for your ATARI)	29,80
173	PD-Program Descriptions (ATARI)	9,80
174	ZX-81/TIMEX Progr. i. BASIC a. Machine Lang.	29,80
176	Programs + Tricks for VIC's	29,80
177	CP/M – MBASIC and the OSBORNE	29,80
178	The APPLE in Your Hand	39,00
182	The Great Book of Games Vol. I - Games f. the C-64	29,80
183	More on the Sixtyfour	39,00
	Riesenprogrammsammlung in BASIC	
8048	BASIC Software Vol. VI	199,00
8049	BASIC Software Vol. VII	159,00
8050	BASIC Software Vol. I	99,00
8051	BASIC Software Vol. II	99,00
8052	BASIC Software Vol. III	149,00
8053	BASIC Software Vol. IV	39,00
8054	BASIC Software Vol. V	39,00

Der Hofacker Verlag produziert und vertreibt neben einer sehr großen Auswahl an Fachbüchern für Elektronik und Microcomputertechnik noch:

- Leerplatinen und Bauanleitungen für Zusatzeinrichtungen für Ihren Personalcomputer, sowie
- Programme (Software) für die bedeutenden Personalcomputer.

(i. V. bedeutet: Buch ist in Vorbereitung)

HOFACKER

HOLZKIRCHEN SINGAPORE LOS ANGELES

ISBN 3-921682-70-3